AFIT/GSO/ENS/91D-03

RADAR SYSTEM CLASSIFICATION USING NEURAL NETWORKS

THESIS

David Michael Cameron
Captain, CAF

AFIT/GSO/ENS/91D-03

AFIT/GSO/ENS/91D-03

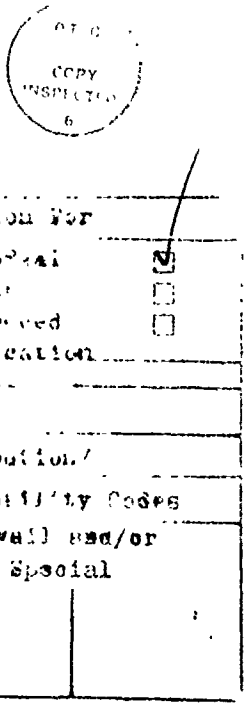
RADAR SYSTEM CLASSIFICATION USING NEURAL NETWORKS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science (Space Operations)

David Michael Cameron, B.A.Sc.
Captain, CAF

December, 1991



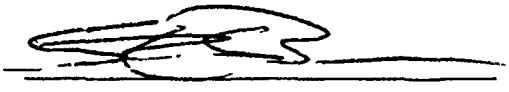
Thesis Approval

Student: Captain David M. Cameron

Section: GSO 91D

Thesis Title: Radar System Classification Using Neural Networks

Defense Date: November 14, 1991

COMMITTEE	NAME/DEPARTMENT	SIGNATURE
Advisor	Major Steven K. Rogers/ENG	
Reader	Major Bruce W. Morlan/ENS	

Preface

This study applied artificial neural networks to the problem of classifying radar emitter systems. The study concentrated on the particular problems associated with classifying radar systems when there are many classes to choose from. Some of the factors affecting the network accuracy were identified and used to improve performance. A method of using multiple neural networks to improve the overall classification accuracy was tested.

I am deeply indebted to my thesis advisor, Major Steven Rogers, for his expert guidance and encouragement. I also wish to thank Captain Daniel Zahirniak for his timely advice and for acquiring the data used in the experiments. The support of Captain Gregory Tarr, the author of the software used for this research, was greatly appreciated. Lastly, I am grateful to Major Bruce Morlan for his editorial input.

David Michael Cameron

Table of Contents

List of Figures

List of Tables

AFIT/GSO/ENS/91D-03

Abstract

This study investigated methods of improving the accuracy of neural networks in the classification of large numbers of classes. A literature search revealed that neural networks have been successful in the radar classification problem, and that many complex problems have been solved using systems of multiple neural networks. The experiments conducted were based on 32 classes of radar system data. The neural networks were modelled using a program called the *Neural Graphics Analysis System*. It was found that the accuracy of the individual neural networks could be increased by controlling the number of hidden nodes, the relative numbers of training vectors per class, and the number of training iterations. The maximum classification accuracy of 96.5% was achieved using a hierarchy of neural networks in which the classes were partitioned based on their performances in a large neural network trained with all classes.

RADAR SYSTEM CLASSIFICATION USING NEURAL NETWORKS

I. Problem Statement

1.1 Background

Electronic warfare equipment is continually evolving and is the focus of much Air Force research. One active area of research is the design of radar warning receivers. The simplest design is one which only indicates the presence of electromagnetic radiation in a specified bandwidth. A more sophisticated design will attempt to identify the radar system which emitted the radar signal, based on the characteristic features of the signal. Success in this type of automatic recognition has been limited by the on-board computing capacity of aircraft and the ever-increasing number of types of radar systems. This area of research is important because the realization of a robust, accurate radar system recognizer would allow instant characterization of the threat to an aircraft.

As described by Ruck (13:5-7), the process of automatic pattern recognition consists of three sequential stages: segmentation, feature extraction, and classification. Segmentation is the operation of isolating the signal of interest from its environment. Feature extraction involves processing the data to compute the features which allow discrimination between different signal classes. The classification stage assigns each input signal to a class based on its features.

Artificial neural networks (also called simply neural networks) have been used extensively to solve problems at all three stages of automatic recognition (1, 6, 13, 18). Neural networks are inherently fast processors due to the parallel processing of information. Neural networks used for classification have been shown to closely approximate statistically optimal performance without requiring statistical analysis of the input data. These characteristics make neural networks more suitable for applications in which processing time is critical.

A major problem with all automatic recognition systems is the d ·rease in system classification accuracy with the increase in the number of distinct classes. As intuition suggests, a system makes more errors discriminating between many objects than between relatively few objects. Radar classification systems are no exception; they are limited in the number of classes that can be recognized. In particular, the application of neural networks to radar system classification is limited by the many-class problem.

1.2 Research Objective

The objective of this research was to determine some of the factors affecting the classification accuracy of a neural network with many output classes and to evaluate methods of increasing that accuracy.

1.3 Chapter Outlines

The following pararaphs contain brief descriptions of the contents of each of the subsequent chapters.

1.3.1 Chapter 2 The literature relevant to the general problem of neural network pattern recognition and the particular problem of radar system classification is reviewed.

1.3.2 Chapter 3 The methodology used to carry out the research is described in detail, including the data, hardware, and software.

1.3.3 Chapter 4 The effects of data and network parameters on the classification accuracy are described. The factors identified are used to improve the performance of networks in subsequent experiments.

1.3.4 Chapter 5 A statistical analysis of the data is presented and the relationship of statistical parameters to network performance is analyzed. The motivation for the use of multiple network systems for classification is developed.

1.3.5 Chapter 6 The performance of various systems of neural networks are evaluated and compared to the performance of single neural networks. Two types of multiple network systems are evaluated: parallel and hierarchical.

1.3.6 Chapter 7 The main conclusions drawn from the results of experiments in chapters 4-6 are summarized. Recommendations for further research are made.

1.4 Scope

The research was limited to a single neural network topology. The two-layer, feed-forward, multi-layer perceptron model trained by a modified conjugate-gradient paradigm was used for all experiments. Since the research was concerned with the relative classification accuracy of neural network models, the use of only one topology was not considered a limitation.

The chapter on statistical analysis is not intended to represent a complete data characterization; it is only intended to identify some of the statistical parameters which relate to the performance of the neural networks.

1.5 Summary

The problem of automatic radar system classification is important to the Air Force. The objective of this research was to determine some of the factors affecting the performance of neural networks in this application. Methods of improving the neural network classification accuracy were investigated. The contents of each of the following chapters was outlined above. The research effort is limited to experimentation with one neural network topology.

The following chapter contains a review of neural networks and current research in the area of radar signal classification.

II. Literature Review

2.1 Introduction

This section reviews literature pertinent to this research. The discussion covers the following three topics:

- Neural network taxonomy
- Radar signal classification by neural networks
- Systems of neural networks

2.2 Taxonomy of Neural Networks

An artificial neural network is a mechanism for performing a mapping from an input vector space to an output vector space. Its structure is analogous to a biological neural network in which a large number of neurons are arranged in some pattern with interconnections between them. The nodes in an artificial neural network correspond to the neurons, and each node is a simple computing element which implements an *activation function*. The strength of the interconnections between nodes are represented by weights which are numbers assigned by a network training process.

Neural networks can have many different forms, and many categorizations are possible. The type of input data, binary or continuous, forms one dichotomy. Kuhl (7) further characterizes neural networks by three properties:

1. Network topology
2. Computational element (activation function)
3. Training algorithm

2.2.1 Network Topology Many topologies are possible, ranging from simple structures to very complex networks of interconnected nodes. However, to be useful a neural network must have an organized and regular structure. The most commonly used structure is the feed-forward network in which the nodes are arranged in layers with each node connected in a forward direction to all the nodes in the adjacent

layer. The number of layers, the number of interconnections, and the number of nodes in each layer are all variable.

There are few guidelines available for choosing the appropriate network topology for a particular problem. It has been shown that a two-layer (one hidden layer) feed-forward neural network can approximate any continuous mapping to arbitrary accuracy (2). The number of nodes required in the hidden layer has not been determined analytically for the general case (17:206). In some applications, a neural network with one hidden layer can outperform a network with two hidden layers (17:207).

2.2.2 Computational Elements Neural networks, when used as classifiers, can be divided into four general groups based on the computing elements and method used for classification. The groups, the corresponding computing elements, and representative classifiers are shown in Table 1 (9:48-49).

Table 1. Basic Classifier Groups

Group	Computing Element	Representative Classifiers
Probabilistic	Distribution Dependent	Gaussian, Mixture
Hyperplane	Sigmoid	Multilayer perceptron, Boltzmann machine
Receptive Fields (Kernel)	Kernel	Potential Functions, CMAC
Exemplar	Euclidean Norm	K-Nearest Neighbor, LVQ

Probabilistic classifiers model the input data as samples from an assumed probability distribution, such as Gaussian. The probability that a particular input vector belongs to a particular class is then proportional to the value of the probability density function for that input. The distributions are usually chosen based on a statistical analysis of data for which the class is known. The performance of the classifier depends on the accuracy of the model (9:47).

Hyperplane classifiers partition the higher dimensional space represented by the input vectors into regions which correspond to the different classes. Typically,

the computing element is a non-linear function such as a sigmoid or a polynomial. These neural network classifiers are characterized by long training times, low memory requirements, and rapid classification (9:49).

Receptive field classifiers use a kernel function which give each node a receptive field in the pattern space; a node responds more strongly the closer the input vector is to its field. These neural network classifiers have relatively short training times (9, 18:49).

Exemplar classifiers compare each input vector to stored examples (or exemplars) of each class and measure the distance between the input and each example. The class of the input vector is indicated by the smallest distance. Neural networks of this type train quickly but may require large amounts of memory and classify relatively slowly, depending on the size of the problem (9:49).

2.2.3 Training Algorithms Neural networks vary greatly in the methods used for training. Training is simply the method used to set the weights of the internodal connections. There are as many techniques for training neural networks as there are neural networks, but all techniques are either supervised, unsupervised, or some combination of supervised and unsupervised (9:48).

Supervised training methods require data which has been labelled with the class it was derived from. Examples of all classes expected as possible input to the trained net must be available. The exact number of exemplars required for effective training has never been determined analytically, but there are approximate formula such as Foley's Rule which requires that "the number of training samples per class should be greater than three times the number of features" (13:30). For each example in a single class, the weights in the network are adjusted according to some rule which results in a better approximation of the desired input/output mapping. Usually, a large number of iterations is required before the neural network is optimally trained.

The most popular supervised training method is the gradient-descent or back-propagation method. It seeks to minimize the squared error between the actual outputs and the desired outputs, resulting in a convergence towards the desired input/output mapping. The multilayer perceptron trained by back-propagation has been shown to approximate the probability functions of the training classes (12). However, the number of nodes required to guarantee a close approximation has not

been analytically determined, and must be found by experimentation for each new application.

Unsupervised training paradigms require no prior knowledge of the input training data. The input vectors are clustered into sets according to their relative positions in the input space. The Kohonen self-organizing feature map is the most common example of this type of training method.

The combined supervised/unsupervised training methods usually begin with unsupervised training to cluster the input data and then "fine-tune" the network with supervised training. This is the preferred method if there is only a small amount of labelled data available, or if the training time must be miminized (9:48).

2.3 Radar Signal Classification by Neural Networks

2.3.1 Advantages of Neural Networks There are many advantages of neural networks over conventional computational methods. Neural networks are very general structures which can be adapted to a wide variety of problems (7). They are also faster than conventional methods, and are more tolerant of system faults or noisy input data (17:200). These advantages are important if a robust, real-time classification system is the objective.

Brown and others (1) reported that a neural network classifier equaled the performance of a conventional statistical classifier in a radar classification application. Zahirniak (18) used radial basis functions (RBF's) as the activation functions for his radar system classifier and reported performance equal to that of hyperplane classifiers. These two examples demonstrate the potential of neural networks in the area of radar signal classification.

2.3.2 Problem Areas Wilson (17:200,209) reported two difficulties in implementing a neural network radar classifier. First, the segmentation task becomes very difficult when there are multiple radar sources. The pulses must be de-interleaved in order to allow separate analysis of the pulses from each emitter. The receiver must measure parameters of the pulses in order to distinguish between pulses from different emitters; in effect, clustering of the input vectors must be carried out. The second difficulty reported was the very long training time required for the backpropagation method.

Howitt (5:213-215) identified two problems in his neural network model of a radar emitter identifier. The first problem was that the network failed to detect when an input pattern belonged to no known emitter class. The second problem was the steep rise in training time required as the number of emitter classes increased.

None of the studies cited specifically reported the problem of decreasing accuracy with increasing numbers of classes; however, none of the systems were tested with more than 10 classes. The many-class problem is often avoided or taken for granted in research but it is a serious concern in the design of real-world systems whether they are based on neural networks or more conventional techniques.

2.4 Systems of Neural Networks

2.4.1 Motivation Complex classification problems are often solved using systems of several neural networks (3, 14, 8). The multiple neural networks can be arranged in parallel, series, or some combination of the two. The series arrangement is normally called a *hierarchical neural network*. The criteria for adopting a particular system is related to the nature of the problem. If the problem can be functionally or logically divided, then each portion can be solved by a separate parallel neural network. If the problem requires multi-level processing of data, then a hierarchy of networks may be more appropriate. The motivation for using a multiple-network system is to achieve greater classification accuracy than is possible with a single neural network by using knowledge about the problem in the design of the system. For instance, a problem in visual model-matching was solved by using a coarse-to-fine strategy which naturally fits into a hierarchical pattern (8:84).

2.4.2 Hierarchical Neural Networks Villa and Reilly (16:657) define a *hierarchical neural network* as follows:

> A *hierarchical neural network* (HNN) is a multi-layered neural network in which the outputs of deeper layers produce progressively partitioned spaces, in which certain functional properties (what the cell "stands for" of cells in a layer are determined by cells in preceding layers. This confers on these nets the property of being partially ordered sets (poset, well represented by a hierarchical diagram).

The advantages of using HNN's are (16:663):

- each layer contains some information about the problem
- Intermediate layer results may be useful even if the system errs
- attributes are inherited and passed on to successive layers

The disadvantages of using HNN's are (16:663):

- criticality of valid output from earlier layers
- a large tree structure
- serial processing by successive layers is slower than parallel

2.4.3 Implementations Ersoy and Hong designed a system consisting of a number of neural networks in series. Each network output included an error detection capability to prevent erroneous information from propagating through the system during training. After a certain number of training iterations, if an input vector produced an output different from the desired output, the vector was transformed by a modified discrete Fourier transform before being used again. This technique prevented the errors of earlier networks from influencing the training of later networks, and also transformed the input vector into a new location in the feature space which was easier to classify. The system reportedly outperformed a 3-layer feed-forward neural network (3:170-174).

Sun, Chen, and Lee designed a system which automatically constructed a hierarchical neural network tailored to the application. The features were ranked based on their information entropy, and the most important features for most classifications were used by the earlier networks. The system was reported to outperform neural networks trained by back-propagation for decision-tree problems (14:491-196).

2.5 Summary

The various neural network topologies, computing elements, and training algorithms have been reviewed. It has been demonstrated that neural networks are capable of performing any continuous mapping and are therefore suitable for many applications. A case was cited in which a neural network equaled the performance of a conventional classifier. The suitability of neural networks for classifying radar signals was confirmed by the positive results of several studies.

Neural networks have three major characteristics which make them especially effective for classification problems. As detailed by Roth (11:36-37), these characteristics are massive parallelism, automatic clustering during learning, and integration of diverse features.

Systems of neural networks have been shown to be more effective in complex applications than single neural networks. The systems should be tailored to match the underlying structure of the problem for the best results.

III. Methodology

3.1 Introduction

This chapter outlines the methodology and tools used to conduct the research. The data, hardware, software, and experimental approach are each described.

3.2 Data Description

The data from the Georgia Technical Research Institute comprises 32 ASCII files representing 32 classes of radar emitter data. The first line of each file contains two integers which give the number of data vectors (one per line) and the length of the vectors (all vectors contain 16 elements). The remaining non-blank lines in the file alternate between 16 integers and a single integer representing the class number of the preceding vector. The number of sample vectors of each class is shown in Table 2.

The vector elements are the measured features of the radar signals. The specifics of the measurements used for features are irrelevant to the classification problem analysis.

The data was used to train and test the various neural network models. The research consisted of experiments to classify the data vectors and analysis of the classification performance. Thus, the data provided a specific problem for a case study as well as a means for evaluating the success of various methods in classifying many classes.

3.3 Hardware

The computers used to run the neural network simulations were Silicon Graphics workstations. The three models used were the IRIS 3120, the IRIS 4D/35, and the IRIS 4D/310GTX. The operating systems were versions of UNIX. Other UNIX-based systems were used to prepare and manipulate the ASCII data files.

3.4 Software

The neural network simulations were all performed using the *Neural Graphics Analysis System* program (15). The program was a powerful tool for research and

Table 2. Original Data Library

File name	Class Number	Number of Vectors
class_01.d	1	250
class_02.d	2	250
class_03.d	3	2183
class_04.d	4	250
class_05.d	5	250
class_06.d	6	400
class_07.d	7	2183
class_08.d	8	84
class_09.d	9	84
class_10.d	10	679
class_11.d	11	870
class_12.d	12	566
class_13.d	13	594
class_14.d	14	571
class_15.d	15	591
class_16.d	16	591
class_17.d	17	594
class_18.d	18	561
class_19.d	19	296
class_20.d	20	592
class_21.d	21	523
class_22.d	22	584
class_23.d	23	593
class_24.d	24	551
class_25.d	25	229
class_26.d	26	96
class_27.d	27	126
class_28.d	28	105
class_29.d	29	591
class_30.d	30	35
class_31.d	31	28
class_32.d	32	796

was used to implement all neural network models. The program version used in this research was the current version on July 27, 1991.

The Neural Graphics program implements a neural network model using a data file containing both training and test vectors. The program provides an exhaustive report of the network output from each node for each test vector including statistics on the accuracy of the network by classes.

The program provides several measures of the accuracy of the neural network in the classification task. It measures the accuracy using both the training and test data, and for each set it calculates the percentage "right" and "good". The program defines a "good" classification as one in which the output node corresponding to the input vector class has the highest output. The program defines a "right" classification as one in which the correct output node value is greater than 0.8 and all other nodes have values less than 0.2. The less strict "good" measure was used throughout this thesis and was termed the percentage *correct*.

Several modifications were made to the program to allow the network models to test and train with large data files with many classes. The minor changes consisted of changing some integer constants in the program modules *makeinput.c* and *test.c*. The author has since updated the program several times and removed the limitations; however, the updates included changes to the training paradigms. In the interest of maintaining a standard basis for comparing experiments, the old (modified) version was used throughout this thesis.

Many programs were written to create and manipulate data files. Initially, the data files had to be reformatted to allow processing by the Neural Graphics program. Since the Neural Graphics program requires sequentially numbered classes beginning at one, a method of changing the class number of vectors was required. In addition, programs to randomly pick vectors of specified classes and to analyze Neural Graphics output files were required. The main C programs and Unix script files used are listed in Appendix D.

3.5 Experimental Design

An initial period of experimentation with the Neural Graphics Analysis System was conducted to determine the proper procedures and formats required by the program, and to decide on a standard network topology. The topology chosen

was a feed-forward two-layer neural network, based on its simplicity and the good results obtained in the initial experiments. The particular topology chosen was not considered as important as the maintenance of a standard for all experiments. The initial experiments were also used to pick appropriate program parameters for the radar system identification problem. Parameters which worked well were chosen and kept constant throughout this research unless specifically noted. The standard parameters of importance are:

- Conjugate Gradient -- the training paradigm used
- Saliency Off -- only used when measuring feature saliency
- Class Output -- normal node to class correspondence: one node per class
- Statistical Normalize -- vectors are normalized by feature
- Random initial weights -- to initialize the neural network for training
- Layers -- 2 layers of weights (number of hidden nodes varies)

Each experiment was conducted with one or more unique data files constructed from the original data. The data files varied in the total numbers of training and test vectors, the particular classes represented, and the numbers of training and test vectors per class. The Neural Graphics program was run for a certain number of training iterations on each data file. These values are all tabulated in the discussion of the particular experiment and the associated tables and graphs.

In comparing the experimental results for various neural networks, the overall network accuracies must be interpreted with respect to the set of test vectors used. A higher overall network accuracy could be caused by the use of a test set with more vectors from classes which the network classifies with high accuracy, and fewer vectors from the more "difficult" classes. This confounding effect was avoided by only comparing results obtained with the same numbers of test vectors from each class.

3.6 Summary

This chapter has outlined the software, hardware, data, and experimental methods used for this research. The GTRI data was used to build data files as

input to the Neural Graphics Analysis System running on Silicon Graphics workstations. The results were analyzed and compared based on the classification accuracy of the networks. The GTRI data served as a case study as well as a means of testing and evaluating the network models. Analysis of the network performances provided insights into both the case study and the general many-class identification problem.

The following chapter contains the results of neural network experiments in which the data and network parameters were varied to determine their effects on the performance of the networks.

IV. Radar Data Characterization

4.1 Introduction

This chapter evaluates the performance of various neural networks in the task of classifying the radar emitter data. The effects of parameters such as the numbers of training vectors, test vectors, hidden layer nodes, and classes on the network accuracy were documented. The effect of features was also investigated. The collection of all the experiments gives a good characterization of how the neural networks perform on this data set, and identifies parameters which are expected to be important in many neural network problems.

4.2 Foley's Rule

Table 3.1 shows the number of vectors of each class. The classes 30 and 31 have 35 and 28 vectors, respectively. According to Foley's rule, there should be a minimum of 48 vectors per class for training the neural network. When this criteria is met, the observed neural network error rate on the training data is a good predictor of the network error rate on an independent set of test data (10:61). Since Foley's rule is an approximate rule-of-thumb, an experiment was conducted to determine if a neural network could train effectively on the classes with few vectors.

Experimental results: see following paragraph

Data file: a100x32.d

Classes: all 32

Training vectors: the lesser of 100 or half of the vectors, for each class

Test vectors: the lesser of 50 or half of the vectors, for each class

Hidden nodes: 20

Iterations: 50,000

The overall classification accuracy was 92.3% for the training vectors and 90.0% for the test vectors. The percentage for the training vectors appears to be a good overall predictor of the network accuracy for the test vectors. Unfortunately, a

class-by-class breakdown of the training accuracy is not available using this Neural Graphics program.

For the test vectors, the classification accuracies for class 30 and 31 exemplars were both 0%. The only other classes with fewer than 48 training vectors were 8 and 9, with 42 each. The network classified those exemplars with 100% accuracy. The accuracies for the other classes varied between 78% and 100% except for classes 2 and 19 at 36% and 48%, respectively.

Since the classes 30 and 31 have too few vectors to effectively train this neural network, they were excluded from all other experiments. These classes were still used in statistical calculations for completeness. Class 32 was renamed class 30 for convenience.

4.3 Baseline Testing

This series of experiments was intended to provide a baseline against which to compare future experimental results. Accordingly, the number of classes per data file and the number of vectors per class were both varied as widely as possible. In addition, the class numbers were reassigned to ensure a random combination of classes.

Experimental results: see Table 3

Data files: named in format *annxmm.d*, where *nn* is the number of training vectors per class, and *mm* is the number of test vectors per class.

Classes: randomly selected, numbers as indicated

Training vectors: as indicated

Test vectors: 25 per class

Hidden nodes: 16

Iterations: as indicated

Special note: some classes have too few vectors: thus the *insufficient data* entries

It can be seen that the classification accuracy increased as the number of training vectors increased. The accuracy decreased as the number of classes increased, as expected. The accuracy generally increased as the number of iterations increased.

and the effect was more pronounced for runs with twenty or more classes. This was also expected since the larger networks had to be trained with more vectors. The maximum accuracy recorded for a 30-class problem was 90.4%.

Table 3. Baseline Testing

Training		Percentage Correct (Test/Train)				
Vectors	Iterations	Number of classes				
(per class)	(x 1000)	8	14	20	26	30
25	10	96.0/98.0	95.7/95.1	84.8/88.4	84.1/82.5	70.0/69.9
25	20	97.0/99.5	96.3/98.3	88.4/91.4	87.7/89.1	87.1/88.9
25	30	97.5/99.5	96.0/99.1	91.0/94.0	89.5/90.8	88.3/89.6
50	10	96.0/98.5	94.3/95.7	90.4/92.1	88.3/90.0	81.5/80.0
50	20	96.0/99.0	94.9/98.3	89.4/91.9	92.3/94.8	90.4/91.7
50	30	95.0/99.5	94.6/98.6	90.8/95.5	91.9/94.9	88.9/91.5
75	10	96.0/97.3	96.6/96.5	86.2/88.2	82.0/83.0	insuf. data
75	20	96.5/98.8	97.4/97.7	89.2/91.7	91.5/93.2	insuf. data
75	30	96.5/99.3	97.7/98.0	91.8/94.2	92.0/92.5	insuf. data
100	10	97.5/99.6	96.0/96.5	87.4/90.2	78.3/79.3	insuf. data
100	20	97.5/100	96.9/96.9	91.6/93.1	89.1/90.0	insuf. data
100	30	98.0/100	96.9/97.6	91.2/94.7	92.0/92.5	insuf. data
125	10	98.0/98.5	95.7/96.3	90.2/90.6	76.0/77.1	insuf. data
125	20	98.0/99.2	96.0/97.5	90.4/91.4	88.3/89.8	insuf. data
125	30	98.5/99.5	96.9/97.6	92.6/94.8	90.9/91.9	insuf. data

4.4 Effect of Number of Classes

This experiment investigated the relationship between classification accuracy and the number of classes to be discriminated.

Experimental results: see Figure 1

Data files: named in format *a50xnn.d*, where *nn* is the number of classes

Classes: randomly selected, variable number

Training vectors: 50 per class

Test vectors: 25 per class

Hidden nodes: 16

Iterations: as indicated by legend (k means "x 1000")

Special Note: accuracies plotted are for test vectors

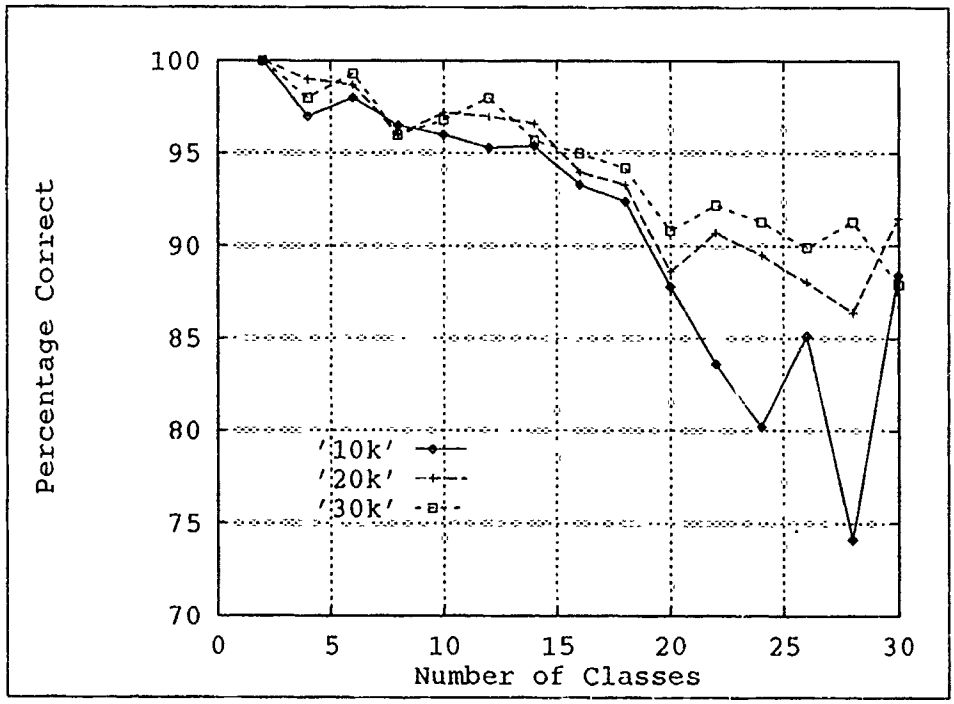


Figure 1. Effect of Number of Classes

The classification accuracy decreased as the number of classes increases, as expected. The decrease was not due to a limitation of the network capacity (only 16 hidden nodes), as demonstrated in section 4.8. The accuracy increased as the number of iterations increased. The graph of Figure 1 clearly shows the many-class problem which hampers automatic identification systems.

4.5 Convergence of Network Accuracy

In some cases, the accuracy of a neural network will only reach its maximum after a large number of iterations. This is usually apparent in networks with a large number of output nodes (meaning a large number of classes). For this experiment, the accuracies of both a 20-class and 26-class network were monitored as they were trained for a large number of iterations.

Experimental results: see Figure 2 and Table 18 (Appendix A)

Data files: a50x20.d and a125x26.d

Classes: 20 and 26, respectively (randomly selected)

Training vectors: 50 and 125 per class, respectively

Test vectors: 25 per class

Hidden nodes: 16

Iterations: on x-axis

Special Note: accuracies plotted are for test vectors

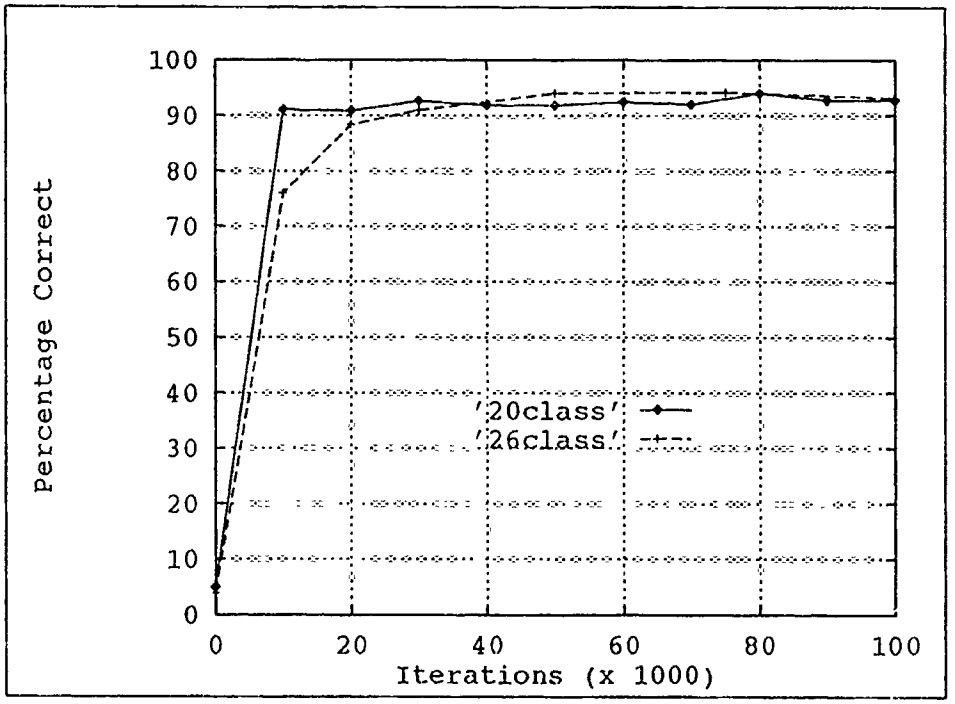


Figure 2. Convergence of Network Accuracy

The convergence of the networks' accuracies occurred within a few thousand iterations for this data. The 26-class problem takes longer to converge because there are more classes and more training vectors per class. Overtraining was not observed: the network classification accuracies did not drop significantly during training to 200,000 iterations.

4.6 Binary-Coded Output Nodes

Neural networks with many output nodes have a large number of weights and therefore much computation is required for training the network. One possible way of reducing the number of output nodes is to assign a coded meaning to each of the output nodes. Instead of the usual class-coded output in which the node with the highest output value signifies the class number, each node could be interpreted as a digit of a binary number. The output 1010 could be interpreted as class 10. In this experiment, the network accuracy was determined for various sizes of networks trained to output the binary-coded class number.

Table 4. Effect of Binary Coding of Output

		Percentage Correct (Test/Train)	
File name	Iterations	Binary Output	Class Output
a50x08.d	10,000	96.5/99.8	96.0/98.5
"	30,000	96.0/99.8	95.0/99.5
a50x14.d	10,000	92.3/93.6	94.3/95.7
"	30.000	95.7/96.7	94.6/98.6
a50x26.d	10,000	67.2/68.5	88.3/90.0
"	30,000	76.9/77.6	91.9/94.9
a125x08.d	10,000	95.5/96.8	98.0/98.5
"	30.000	95.5/98.6	98.5/99.5
a50x20.d	10.000	79.6/81.4	90.4/92.1
"	30.000	76.4/79.3	90.8/95.5
"	60.000	86.0/90.1	n/a
"	90.000	85.6/89.6	n/a
a125x20.d	10.000	82.0/82.0	90.2/90.6
"	50.000	78.0/79.0	n/a
"	100.000	82.2/85.4	n/a
"	125.000	83.8/85.8	n/a
"	200.000	78.4/80.6	n/a

Experimental results: see Table 4

Data files: named in format *annxmm.d*, where *nn* is the number of training vectors per class, and *mm* is the number of classes.

Classes: randomly selected: 8, 14, 20, or 26

Training vectors: as indicated by file name

Test vectors: 25 per class

Hidden nodes: 16

Iterations: as indicated

Special notes: Neural Graphics output parameter set to *Binary Output*. last column of Table 4 was taken from Table 3

The binary-output neural network performed marginally better than the equivalent class-output neural network only for the 8-class, 50-vectors-per-class data set. In networks with 14 or more classes and networks with 125 training vectors per class, the binary-output network performed relatively poorly. The binary-output networks took longer to train on this data. The binary coding scheme performed progressively more poorly as the number of classes increased and/or the number of training vectors increased. For the 20-class, 125-vectors-per-class data set, the class-output network performed 6.4% better than the binary- output network, even though the latter was trained to its maximum accuracy (which occured at 125,000 iterations). The binary coding of the output nodes generally decreased the accuracy of the neural networks.

4.7 Repeatability of Experiments

The consistent results of the previous experiments seem to indicate that the accuracies obtained are repeatable and are independent of the initial random weights. This experiment was run to determine explicitly if the results obtained using Neural Graphics are repeatable. Three data files were run a total of 40 times, starting each time with a new random set of weights. Both normal class-coded output and binary-coded output were used (once on the same file).

Experimental results: see Table 5

Data files: a50x14.d, a75x20.d, and a50x26.d

Classes: 14, 20 and 26, respectively; randomly selected

Training vectors: 50, 75 and 50 per class, respectively

Test vectors: 25 per class

Hidden nodes: 16

Iterations: 20,000 each run

Special Note: The last two rows represent runs with the Neural Graphics ouput setting at "Binary".

Table 5. Repeatability of Experiments

		Percentage Accuracy (Test/Train)	
File name	Runs	Averages	Deviations
a75x20.d	10	90.1/92.6	.87/.86
a50x14.d	10	97.0/97.7	.80/.26
a50x14.d	10	90.0/95.1	1.68/.29
a50x26.d	10	72.2/74.2	2.13/1.62

The standard deviations were less than 1% for the runs with normal class-coded outputs, and were less than 2.2% for the runs with binary-coded outputs. The deviations were smaller for networks with the normal class-coded outputs than with the binary-coded outputs. The deviations were larger for networks with more classes.

The standard deviations are expected to be even smaller for runs made with more iterations, due to the continued convergence of the neural network training which occurs after 20,000 iterations. The results of the Neural Graphics runs can be considered independent of the initial weights when they are randomly set. The results are repeatable within a small statistical variation.

4.8 Effect of Number of Hidden Layer Nodes

The number of hidden layer nodes required to optimize the classification accuracy of a neural network has not been determined analytically for the general case. This experiment investigated the relationship between network performance and the number of hidden layer nodes. The goal was to find the minimum number of nodes required to ensure good classification performance.

Experimental results: see Figure 3 and Table 19 (Appendix A)

Data files: as indicated in Table 19

Classes: as indicated

Training vectors: indicated by 1st number in file name

Test vectors: 25 per class

Hidden nodes: as indicated

Iterations: 20,000 for Figure 3; as indicated in Table 19

Special note: the program sometimes crashed when the number of hidden nodes was small and the total error grew very large

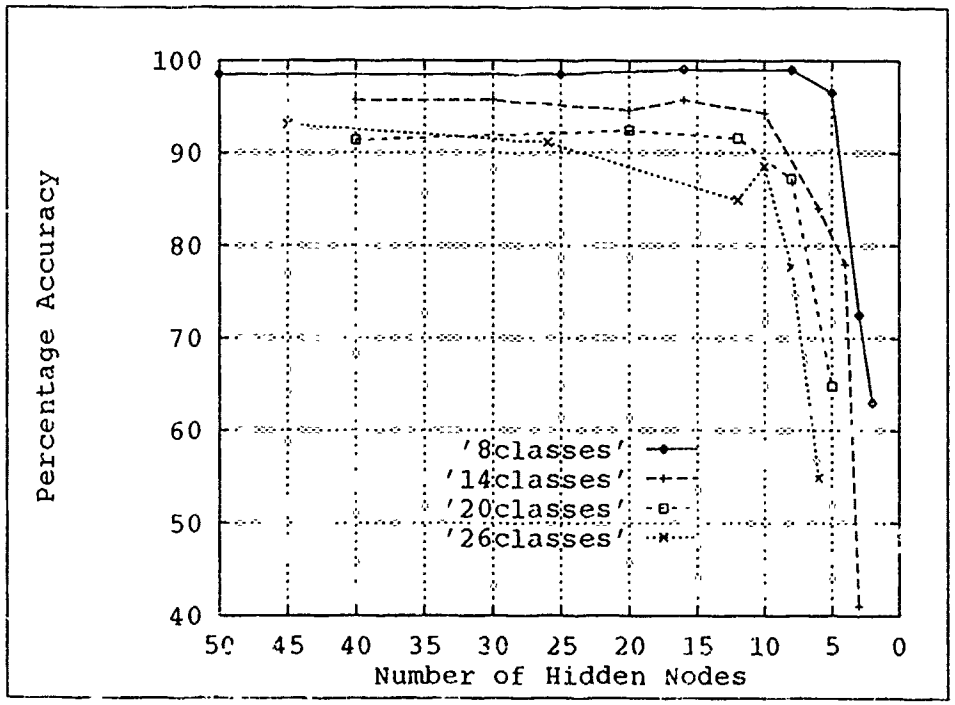


Figure 3. Effect of Number of Hidden Nodes (20,000 Iterations)

The performance of the neural networks was not increased (and the training time was greatly increased) by using more hidden nodes than the number of classes. The performance degraded when the number of hidden nodes was less than about one-half the number of classes. For this data set, optimum classification performance can be ensured by using more hidden layer nodes than half the number of output classes.

4.9 Effect of Relative Numbers of Training Vectors

All 16,939 vectors of the 30 usable classes were divided into two equal sets of vectors. One of the sets was designated training vectors, and the other was designated test vectors. A 30-class neural network was constructed and trained with varying numbers of training vectors as follows:

- alldata.d: all trng vectors used
- max100.d: the lesser of 100 or all the trng vectors per class
- all100.d: all classes had 100 trng vectors
- some200.d: same as all100.d except classes 1, 2, and 19 had 200 trng vectors each
- some300.d: same as all100.d except classes 1, 2, and 19 had 300 trng vectors each

Classes 1, 2, and 19 were chosen because the 30-class networks had the lowest accuracy of classification for vectors of those classes.

Experimental results: see Figure 4 and Table 20 (Appendix A)

Data file(s): alldata.d, max100.d, all100.d, some200.d, and some300.d

Classes: 30

Training vectors: 100 per class except as indicated above

Test vectors: 8469 total (half the vectors for each class)

Hidden nodes: 20

Iterations: 70,000

Special notes: for the last 3 files, some duplication of trng vectors was required to generate the required numbers, only the first two data files are used for the graph.

The classes with many vectors performed much better when they were allowed a disproportionately large number of training vectors. This shows that those classes had a disproportionately large effect on the training of the neural network. Conversely, the classes with few vectors performed poorly when there were classes with

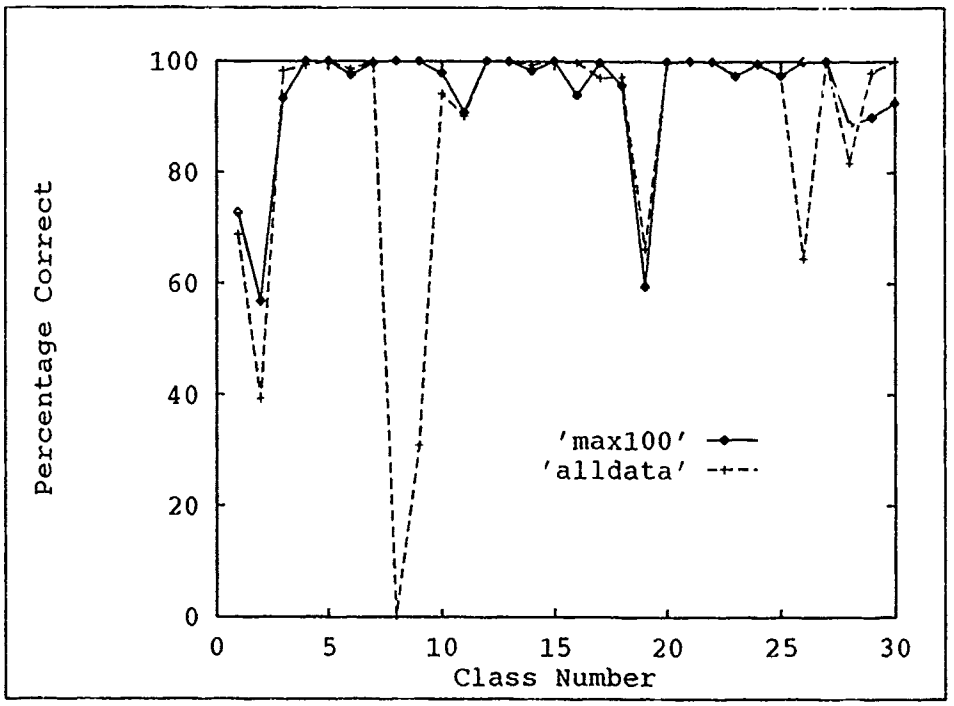


Figure 4. Comparison of All or 100 Training Vectors

many training vectors included in the data set, yet performed very well when the number of training vectors was limited to 100 per class.

The effect of adding more training vectors to the classes with fewer than 100 exemplars was to decrease the network performance slightly on some other classes. When, in addition, the number of vectors in classes 1, 2, and 19 was doubled, only the network performance on class 19 was significantly increased. When the number of vectors for the same three classes was increased to 300 each, the performance of the network on class 19 again increased, with little effect on classes 1 or 2.

The overall accuracies listed in Table 20 are higher than for previous 30-class networks because there are more vectors of some classes which were very accurately classified. The actual performance of the networks on new data would be similar only if the classes were presented in the same proportions (as per Table 2).

It is apparent that there is an optimum number of training vectors for each class which would result in maximum overall classification accuracy. The large number of permutations possible with this 30-class problem prohibits an exhaustive search. In general, training the network with approximately equal numbers of vectors from

each class resulted in good overall accuracy without badly degrading the recognition of any particular classes.

4.10 Effect of Vectors with Equal Feature Values

There are three classes which contain vectors with all 16 features of equal value. The value of the features varies widely between vectors of the same class. These "constant" vectors may be degrading the accuracy of the networks. The classes 1, 2, and 8 have 13, 90, and 11 "constant" vectors, respectively.

Several approaches were used in this experiment. The three classes 1, 2, and 8 were first treated as a 3-class problem (file: const3.d). Then, the constant vectors were eliminated from the data file and the 3-class network was trained again (file: noconst3.d). Then, the 3-class network was trained with only the constant vectors for training and only non-constant vectors for testing (file: allconst3.d). The effect of the constant vectors on the accuracy of a 20-class network was tested by including the classes 1, 2, and 8 (file: const20.d), and then repeating the training with the same classes with the constant vectors removed (file: noconst20.d).

Experimental results: see Table 6

Data files: as indicated

Classes: classes 1, 2, and 8 for 3-class network; other 17 classes chosen at random for 20-class network

Training vectors: 50 per class

Test vectors: 25 per class

Hidden nodes: 16

Iterations: as indicated

The effect of removing the "constant" vectors from the data files was to improve the performance of the neural network on test vectors by 1.8% for the 3-class problem, and by 2.6% for the 20-class problem. The poor performance of the network in the test with all constant training vectors indicates that there is a fundamental difference between these vectors and the other vectors of classes 1, 2, and 8. The better overall performance of the 20-class network is partly due to the better performance of the 17 classes with no constant vectors. These results indicate that the vectors with equal values are detrimental to the classification process.

Table 6. Effect of Vectors with Equal Feature Values

		% Accuracy (Test/Train)			
		Iterations (x 1000)			
File name	Classes	10	20	30	80
const3.d	3	88.0/98.0	88.0/99.3	89.3/99.3	86.7/99.3
noconst3.d	3	93.2/100	91.5/100	93.2/100	91.5/100
allconst3.d	3	38.2/92.0	40.5/95.4	41.3/96.6	41.5/96.6
const20.d	20	87.8/91.1	88.6/93.6	90.8/94.7	90.6/94.4
noconst20.d	20	90.0/93.1	91.2/94.9	90.4/95.6	93.2/97.7

4.11 Network Performance Using Fewer Features

In order to directly measure the contribution of individual features to the classification process, features were systematically deleted from a data set and the performance of the neural network with fewer and fewer features was recorded. Features were deleted in reverse order by arbitrary choice (feature 16 deleted first).

Experimental results: see Figure 5

Data file: c50x30.d

Classes: 30

Training vectors: 50 per class

Test vectors: 25 per class

Hidden nodes: 20

Iterations: 50,000

Special note: the features are counted from the start of the data vectors (e.g. 12 features indicates the 1st 12 features)

There was a graceful degradation of network performance as the number of features was reduced. The higher-numbered features had less of an effect on the network accuracy, but this may have been because each feature became more important as the total number of features was reduced and there was less redundancy. The graph in Figure 5 shows that no feature is much more important to the classification process than any other.

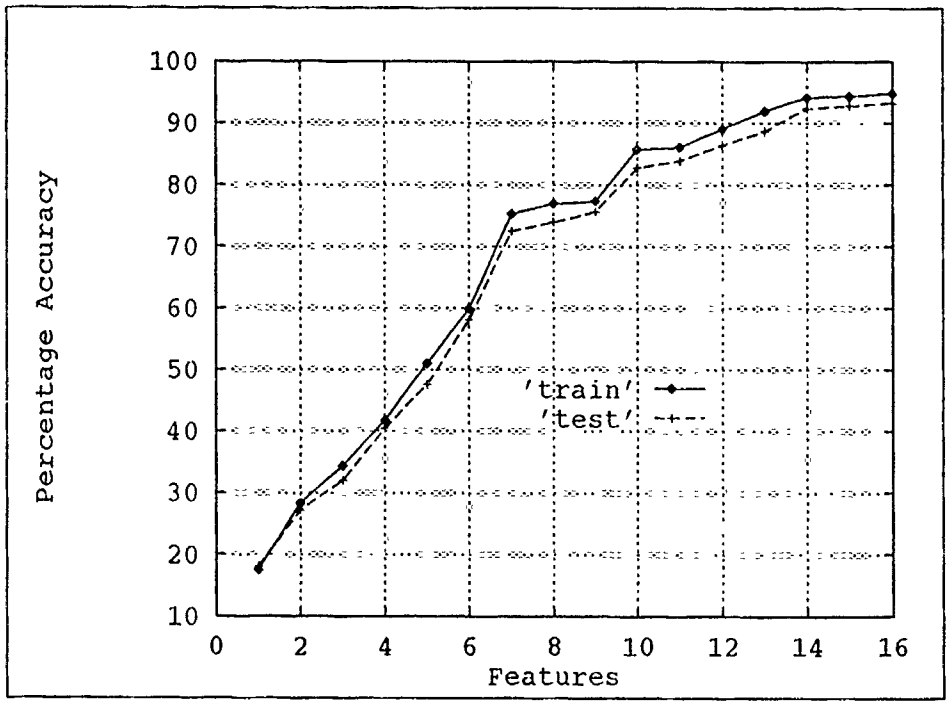


Figure 5. Performance Using First N Features

4.12 Feature Saliency

The Neural Graphics program provides a measure of the usefulness of a particular feature in the network classification process. It is called the *saliency*.

Five data sets were used in 25 runs to determine the saliency of each feature. The average ranking of each feature over all the runs and the final ranking was calculated.

Experimental results: see Table 7

Data files: new1.d, new2.d, new3.d, new4.d, and new5.d

Classes: 30

Training vectors: 50 per class

Test vectors: 25 per class

Hidden nodes: 20

Iterations: variable; 10,000 to 30,000

Special note: Neural Graphics toggled *Saliency On*

Table 7. Feature Saliency Ranking

Feature	Avg. Rank	Ranking
1	4.4	2
2	11.2	12
3	12.6	13
4	10.5	11
5	8.3	9
6	7.1	6
7	7.5	8
8	5.6	5
9	5.0	4
10	7.2	7
11	12.8	14
12	14.0	15
13	14.2	16
14	10.1	10
15	4.5	3
16	1.0	1

Feature 16 was by far the most important feature for the classification process. However, the graph of Figure 5 does not show any significant drop in network accuracy when feature 16 is omitted. This must be due to the redundancy of information contained in the other 15 features. Other features were considerably less important with no large discontinuities in the ranking. The saliency measure was compared to statistical parameters in chapter 5.

The 5 highest-saliency features, the 5 lowest-saliency features, and the 11 lowest-saliency features were extracted from the vectors of the file *new3.d*. The resulting files were used to train neural networks with the appropriate number of input nodes and the resulting performances were recorded.

Experimental results: see Table 8

Data files: best5f.d, worst5f.d, and worst11f.d

Classes: 30

Training vectors: 50 per class

Test vectors: 25 per class

Hidden nodes: 20

Iterations: as indicated

Table 8. Effect of Feature Saliency

File Name	Iterations	% Test	% Train
best5f.d	80,000	87.6	90.3
worst5f.d	50,000	73.9	75.2
worst11f.d	50,000	91.7	93.8

The network using the best 5 features performed 13.7% better (on the test vectors) than the network using the worst 5 features. The worst 11 features produced a trained network with greater accuracy than the network trained with the best 5 features. All features were important to the classification process.

4.13 Summary

The data from class numbers 30 and 31 was excluded from further experiments because they have too few exemplars to effectively train a neural network. The general results of the baseline testing were that the classification accuracy of a neural network decreases as the number of classes increases. The accuracy increases as the number of training iterations increases or the number of training vectors per class increases. The convergence of the network models to a stable value of classification accuracy, and the repeatability of those results, was confirmed. The effect of the binary-coding scheme at the output nodes of the neural networks was negligible when there were only eight output classes, but caused a decrease in the classification accuracy when there were 14 or more output classes.

The networks could be trained close to their maximum accuracy as long as the number of hidden layer nodes remained above half the number of output classes. The anomaly of data vectors with all-equal feature values was shown to decrease the performance of networks. The effect of using many more training vectors of some classes relative to other classes adversely affects the accuracy of the networks on vectors of the smaller classes. When the number of training vectors is limited to a maximum of 100 per class, no adverse effects were noted. The classification of particular classes was improved by including more training vectors of those classes,

but the effect on the classification of other classes was unpredictable. Finally, the 16 features were ranked in importance by the *saliency* measure, but all features were required for the maximum network accuracy.

The following chapter presents a statistical analysis of the data and relates the statistical parameters to the performance of the neural networks.

V. Statistical Analysis

5.1 Introduction

This chapter contains a statistical analysis of the data, including all 32 classes. The effects of parameters such as means, standard deviations, and class distinction on neural network performance were evaluated. The results of some experiments of chapter 4 were related to the observed data distributions.

5.2 Feature Means and Standard Deviations

The entire data library of 17002 vectors was used to calculate the mean and standard deviation of each feature, independently of the class. The results are shown in Table 9.

Table 9. Feature Means and Standard Deviations

Feature	Mean	Deviation
1	139.3	38.5
2	147.6	46.1
3	154.0	48.7
4	156.9	46.6
5	155.6	41.1
6	151.5	34.5
7	146.5	29.3
8	142.3	27.2
9	140.0	27.6
10	139.0	27.5
11	138.4	27.3
12	137.3	25.9
13	136.3	24.7
14	134.8	23.7
15	133.0	22.6
16	130.4	21.9

The feature means vary between 130 and 157; the standard deviations vary between 21 and 49. There are no features which are significantly different in magnitude from all the others. The features appear to be numbered in order of descending mean and standard deviation, except for the first three.

The higher feature numbers might be expected to have less value in classification since they are closer in value across classes, meaning the classes are more similar. However, the saliency measure of section 4.12 ranked features 16 and 15 as first and third, respectively. Feature 1 was ranked second in saliency, yet has an average feature deviation across classes. No clear correlation exists between the standard deviations of the features and their saliency measures.

5.3 Feature Means and Standard Deviations by Class

All 17002 vectors of length 16 were analyzed to determine the means and standard deviations of the features by class. The feature means are shown in Tables 21 and 22 of Appendix B. The feature standard deviations are shown in Tables 23 and 24 of Appendix B.

The means of the first two features for each class were used to create a scatter plot of the projections of the class means into two dimensions (see Figure 6). The plot consists of 32 points representing the means of the features 1 and 2 for each class. The plot shows the positive correlation between the first two features. All sequential pairs of features (i.e. 3 and 4, 5 and 6, etc.) are distributed in a very similar pattern. Random pairs of features (e.g. 3 and 14) showed a much weaker correlation (see Figure 7).

5.4 Effect of Feature Correlation

An experiment was conducted to test the effect of the feature correlations noted in the previous section. From the file *c50x30.d*, three data files consisting of vectors with only 8 features were constructed. The file *f08.d* used the first 8 features; the file *f8even.d* used the even-numbered features; and the file *f8odd.d* used the odd-numbered features.

Experimental results: see Figure 10

Data files: f08.d, f8even.d, and f8odd.d

Classes: 30

Training vectors: 50 per class

Test vectors: 25 per class

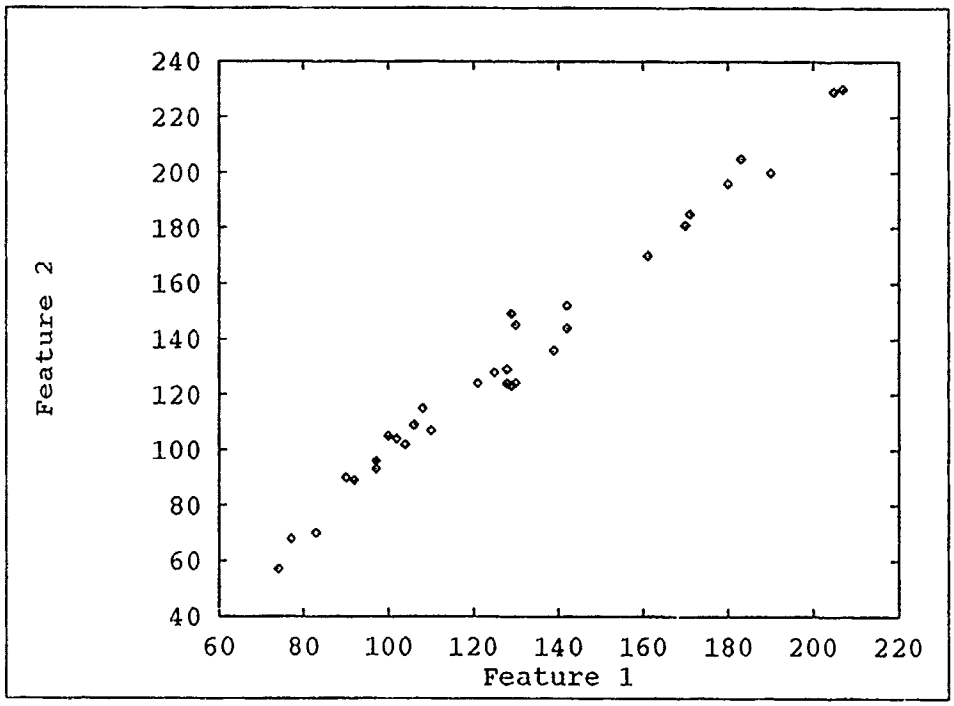


Figure 6. Projections of Class Means (Correlated)

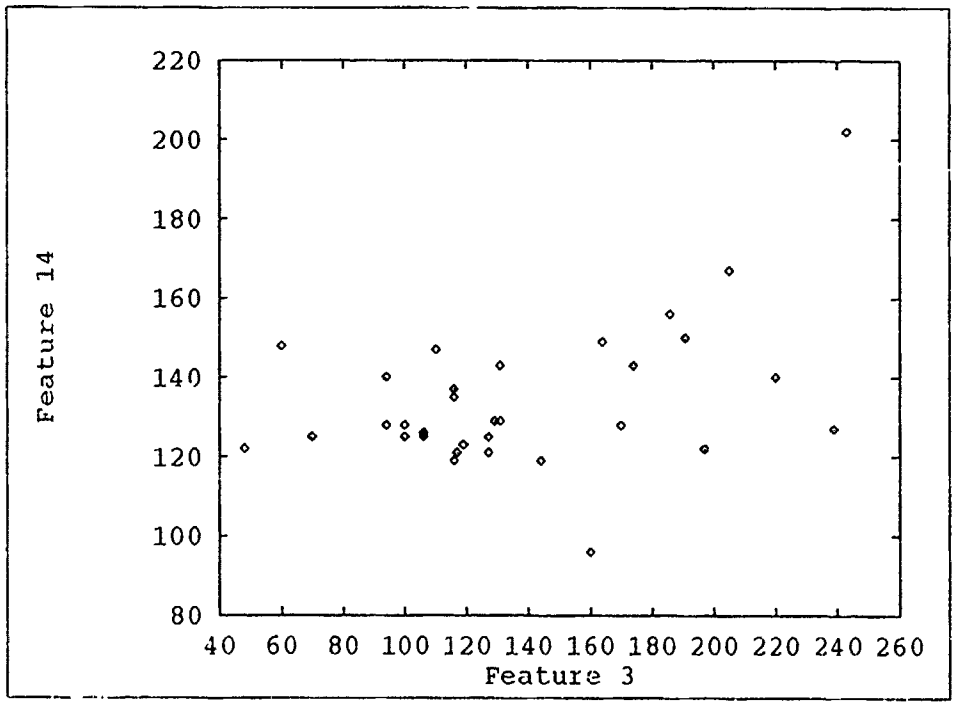


Figure 7. Projections of Class Means (Uncorrelated)

Hidden nodes: 20

Iterations: 50,000

Table 10. Effect of Feature Correlation

File name	% Test	%Train
f08.d	74.0	77.1
f8even.d	92.0	93.1
f8odd.d	91.6	92.9

The first file consisted of 4 pairs of correlated features. The last two files used features which were not clearly correlated. The poor relative performance of the network using the first data file shows that the correlation of the consecutive pairs of features represents a redundancy of information.

The experiments of section 4.11 did not reveal this correlation, but the small drops in classification accuracy of the networks as features were deleted indicate redundant information in the features. Those experiments showed that adding features which were correlated to the existing features still improved the network performance.

5.5 Total Standard Deviation

The class feature standard deviation was used to calculate the total standard deviation across all 16 features. The total deviation is defined here as the square root of the sum of the feature variances, for each class. Table 25 (Appendix B) shows the ranking of the classes by total standard deviation.

Various numbers of classes with the lowest and highest total deviations were selected from Table 25 to construct data sets. The top 10, 12, 14 and 20 classes were used as input to the Neural Graphics program. The bottom 3, 5, and 10 classes (with the highest total deviations) were also tested.

Experimental results: see Table 11

Data files: as indicated; descriptive names

Classes: selected by rank from Table 25

Training vectors: 50 per class

Test vectors: 25 per class

Hidden nodes: 16

Iterations: as indicated

Special Note: the run was stopped if training accuracy reached 100%

Table 11. Effect of Total Deviation on Classification Accuracy

File name	Iterations	% (Test/Train)
lowdev10.d	20,000	100/100
lowdev11.d	20,000	100/100
lowdev12.d	50,000	99.9/100
lowdev16.d	60,000	99.3/100
lowdev20.d	100,000	97.0/98.5
hidev3.d	80,000	98.7/99.7
hidev5.d	80,000	96.8/99.2
hidev10.d	100.000	95.6/99.4

Table 11 shows that 100% accuracy on the test vectors was achieved for the 11 classes with the lowest total deviation. The 10 classes with the highest total deviations lead to a network accuracy of 95.6% on the test vectors. The results show that the total deviation is negatively correlated with the classification accuracy achievable with that class. The high accuracies achieved with the 10 classes with the highest deviations shows that the total deviation alone is not a good predictor of the neural network classification performance. However, since high accuracies (over 90%) are achieved for almost all class groupings, even a gain of a few percentage points is significant.

5.6 Network Trained on Class Means

The distribution of the classes in the 16-dimensional feature space determines how well the classes can be discriminated. If the classes are clustered in non-overlapping groups, one would expect a classifier to perform very well. Conversely, if exemplars of one class are interspersed with exemplars of another class, one would expect poor performance from a classifier.

The simplest distribution of exemplars is the existence of a single cluster center for each class. For this experiment, the center of each class cluster (the class mean) was assumed to be the vector with components equal to the mean of each feature (for that class). The class means were duplicated and used to train a 30-class network.

Experimental results: see following paragraph

Data file: mean30.d

Classes: 30

Training vectors: 5 per class, all equal to class means

Test vectors: 3469 (half of each class)

Hidden nodes: 20

Iterations: 35,000

The neural network trained to a maximum accuracy of 85.2% on the test vectors. The training vectors were learned to 100% accuracy, which was expected since there was only one duplicated vector representing each class. The maximum classification accuracy occurred at 35,000 iterations; further training caused a slight decrease in accuracy after that point. The decrease was 3.0% and was gradual over the subsequent 65,000 iterations.

The performance of the network was very high, considering the extreme compression of data represented by the class averaging. This result implies that the classes must be predominantly clustered about single class centers.

5.7 Distinction of Classes

Based on the class means calculated in the last section, a study of the interaction or overlap of the class distributions was undertaken. The *distinction* measure used here is given by Equation 1: the distinction of any two classes is the sum over all 16 features of the difference of the class means divided by the sum of the class deviations. The means and deviations used were those given in Tables 21, 22, 23, and 24 (Appendix B).

$$D = \sum_{i=1}^{16} |m_{ji} - m_{ki}|/(\sigma_{ji} + \sigma_{ki}) \text{ for classes j, k} \qquad (1)$$

The distinction of every class relative to every other class is given in Tables 26, 27, 28, and 29 (Appendix B). The validity of the single cluster per class assumption can now be tested by building data sets using classes either close together or far apart and training a neural network with them.

Based on the distinction between classes, class numbers 1, 2, 6, 28, and 32 should be difficult to distinguish. A data file consisting of vectors from these five classes was run on Neural Graphics and after 60,000 iterations the classification accuracy was 94.4% and 97.0% for the test and training vectors, respectively. This is relatively poor performance for only 5 classes. By contrast, the six classes 4, 5, 8, 12, 21, and 22 are among the least distinct by the distinction measure. A data file consisting of vectors from these classes trained a network to 100% accuracy on the test vectors after 20,000 iterations.

The distinction of data classes shows a positive correlation with the classification accuracy of networks trained on those classes. Although the difference in accuracy between the two networks discussed in the previous paragraph is only 5.6%, it is significant because it is the *last* 5.6% required for perfect classification.

The distribution of the data is probably unimodal for most classes. Some of the classes have more complicated distributions. There are methods for determining the number of modes in data clusters; the interested reader can refer to Gnanadesikan (4:58).

5.8 Statistical Measures and Class Performance

The statistical measures of total feature deviation and distinction are closely related to the classification accuracy of the neural networks by class.

The classes which were very distinct from other classes by the distinction measure (low values) were ranked among the highest by the total feature deviation measure (see Table 25). Classes 1, 2, 6, and 28 ranked 27, 29, 30, and 25, respectively, out of the 30 classes. The classes which were not very distinct from other classes ranked among the lowest by the total feature deviation measure. Classes 4, 5, 8, 21, and 22 ranked 1, 1, 2, 11, and 8, respectively, out of the 30 classes. A very strong negative correlation between the distinction and the total feature deviation is indicated.

Both of these measures are also strongly correlated with the classification accuracy by class. The classes which had low measures of distinction and high measures

of total feature deviation were also the ones which were classified the least accurately, as indicated by Table 20. Classes 1, 2, and 28 were ranked 29, 30, and 27, respectively, for the network trained with the data file *max100.d*. The classes which had low measures of total feature deviation and high measures of distinction ranked very highly by accuracy.

Therefore, the classification accuracy of the neural networks is also providing statistical information about the classes.

5.9 Summary

The data vectors are fairly well distributed in the 16-dimensional feature space, which explains the fairly high classification accuracies of neural networks evaluated in the last chapter. There is a positive correlation between sequential pairs of features. A neural network trained on only vectors which are equal to the class means achieved 84.3% accuracy on test vectors.

The total deviation of classes and the distinction of a group of classes can both be used to pick groups of classes which are easier to classify than those picked at random. These two statistical measures are strongly correlated with the performance of a neural network. The improvements in classification performance were only a few percentage points, but they were significant because the accuracy was already very high.

The abililty to choose class groupings which allow better neural network performance suggests that an approach using multiple neural networks may improve the overall performance of the classification system. Separate neural networks could be used to classify groups of classes chosen for their high performance, and the group results could be combined to get a complete classification system. The next chapter presents experiments with systems of multiple neural networks.

VI. Multiple Neural Networks

6.1 Introduction

The experiments of this chapter involved systems of neural networks which together formed classification systems. The criteria for grouping the classes was based on the classification performance of a neural network on all of the classes. This differs from the statistical approaches of the last chapter in that only the actual classifications performed by the neural network were used; no characteristics of the data were used.

The general approach was to train a network to divide a large problem into several smaller problems. First, several neural networks were used in parallel. Then, a hierarchical approach was used with neural networks working sequentially.

6.2 Classification Performance

The classification accuracy of a trained neural network was analyzed on a class-by-class basis and the network output and classification of every test vector was analyzed and counted by the program *reperrunch.c* (see Appendix D). The results were tabulated by the program *confuse.c* (see Appendix D) and a frequency table (or confusion matrix) of which input class vectors were mapped to which output classes was constructed. Analysis of this matrix revealed which classes were difficult to classify correctly and which class combinations were frequently confused.

Included in the program was an analysis of the outputs from each node of the neural network for each input vector. The last row of each table, labelled "Not top 3", gives the total errors for which the correct node output was not one of the top 3 in magnitude.

Experimental results: see Tables 30, 31, and 32

Data file: max100.d

Classes: 30

Training vectors: 100 per class; classes 8,9,26,27 have 42,42,18,63, respectively

Test vectors: 8169 (half of each class)

Hidden nodes: 20

Iterations: 70,000

Special Note: the last column of Table 32 includes the errors for the same rows of Tables 30 and 31

6.3 Groupings Based on Inter-class Confusion

The confusion matrix gives the performance of the 30-class neural network with respect to each class in two ways:

1. The number of times a given class is mistaken for any another class. This number is given by the "Errors" row at the bottom of the tables.
2. The number of times all other classes are mistaken for a given class. This number is given by the far right column of Table 32.

The classes were ranked based on their input vector classification accuracy as tabulated in Tables 30, 31, and 32. The ranking is shown in Table 33 (Appendix C). Twelve classes were classified perfectly, and only 5 were classified with less than 90% accuracy.

The classes were also ranked based on the number of times any vector was mistaken as a vector of the class under consideration, as tabulated in the last column of Table 32. This ranking is shown in Table 34. Three classes had no vectors wrongly assigned to them, and 15 classes had fewer than 10 vectors wrongly assigned to them.

Table 33 was used to select the best 15 classes and the worst 6 and 10 classes. Table 34 was used to select the most-confused 12 classes and the least-confused 11 classes. Data files were constructed based on these class groupings for the following experiment (the files can be identified by the number of classes).

Experimental results: see Table 12

Data files: as indicated

Classes: as indicated

Training vectors: the lesser of 100 and half the total per class

Test vectors: 8469 (half the total per class)

Hidden nodes: 16

Iterations: as indicated

Table 12. Class Groups with High and Low Inter-cl. Confusion

File Name	Classes	Iterations	%Test	%Train
worst6.d	6	80,000	90.7	97.0
worst10.d	10	60,000	94.4	99.4
best15.d	15	40,000	100	99.9
nonconf.d	12	75,000	99.1	99.2
conf11.d	11	75,000	96.3	97.1

Since the classes were picked based on the performance of a 30-class network, it is not surprising that the same relative performance carried over to the networks with fewer classes. The result for the file worst6.d is the worst performance of any 6-class problem attempted so far in this research. The best-performing 15 classes scored 100% on the test vectors.

The groupings based on the number of incorrect classifications *into* a class did not result in as dramatic a range of network performance. The least-confused 12 classes scored 99.1% and the most-confused 11 classes scored 96.3% on the test vectors. Therefore, the input classification accuracy was judged a better criteria to use for selecting class groupings and was the criteria used in the later multi-network experiments.

6.4 Output Node Values

The confusion matrix shows that when the network made an error in classification, often the correct node output was not even in the top 3 in magnitude. There were 160 out of 407 errors in which the correct node output was lower than third in magnitude. The program was modified to check the top 5 output nodes, and 121 out of 407 errors were not in the top 5 in magnitude.

The significance of the "top 5" is that very few output nodes have a high output for any given input. The neural networks train to output only one high value and all other values low. Of 100 randomly selected sets of output nodes (1600 nodes), there were only 169 nodes with output values above 0.2. No set had more than 3

values above 0.2. This was true for both correct classification outputs and errors. Therefore, when the correct node output is not in the top 5, it is almost certainly below 0.2 in value and is indistinguishable from the other low node values. The analysis of node output values did not provide any information on the correctness of the indicated classifications.

6.5 Parallel Neural Networks

The initial approach was to use two 15-class neural networks to classify each input vector; the classification corresponded to the highest output node from either network. The classification accuracy was very low because the networks responded unpredictably to vectors from classes it had not been trained with. The outputs of nodes from both networks were analyzed in an unsuccessful attempt to determine which network should be used (the network which was trained on the particular input vector). A method of identifying vectors which do not belong to any class of a given network was required.

Two separate 16-class networks were used to classify each input vector. The first network consisted of the top-performing 15 classes from Table 33 and a composite class made up of 20 vectors from each of the other 15 classes. The second network consisted of the worst-performing 15 classes from Table 33 and a composite class made up of 20 vectors from each of the other 15 classes.

Experimental results: see Table 13

Data files: as indicated

Classes: as indicated

Training vectors: the lesser of 100 and half the total per class

Test vectors: 8469 total (half of each class)

Hidden nodes: 16

Iterations: as indicated

The overall performance of the two parallel 16-class networks on the 30 input classes was 93.5%. This was worse than the performance of the single 30-class network and is therefore not a useful technique for this problem. The values of the

Table 13. Parallel Network Performance

File Name	Classes	Iterations	%Test	%Train
good15.d	15	110,000	99.9	100
bad15.d	15	70,000	95.6	96.2
good16.d	16	70,000	95.6	96.2
bad16.d	16	65,000	92.2	91.1

outputs of nodes of both networks for the same input vector were compared in an attempt to determine the appropriate network to use for classification. The highest output of the "wrong" network was higher than the highest output of the "right" network approximately half of the time. Analysis of the weights did not reveal any method of choosing the right network.

6.6 Hierarchical Approach

The classification accuracy recorded for the individual classes in a 30-class network was used to divide the 30 classes into smaller groups. The overall classification accuracy of the system is the probabalistic combination of the separate neural networks. Three networks will be required for classification:

1. a 2-class network to separate the 30 classes into two groups
2. a network to classify the classes of group 1
3. a network to classify the classes of group 2

The 30 classes were split into two groups in four different ways. First, the classes were split into two groups consisting of the first 15 and last 15 classes, by number. This grouping was used for comparison since it is pseudo-random with respect to class performance. Based on the ranking of Table 33 (Appendix C), the 30 classes were split three other ways: the top 21, 18, or 15, and the bottom 9, 12, or 15, respectively. The class groups were numbered 1 and 2. The number of iterations was chosen for the maximum network accuracy; a further 20,000 iterations beyond the listed values caused either no change or a slight decrease in the accuracy, in all cases.

Experimental results: see Table 14

Data files: as indicated

Classes: 2

Training vectors: the lesser of 100 or half the total, per class

Test vectors: 8469 total (half of each class)

Hidden nodes: 16

Iterations: as indicated

Table 14. Groupings into Two Classes

File name	Iterations	% Group 1	% Group 2
1st2nd15.d	130,000	93.9	97.1
top21bot9.d	110,000	98.1	94.2
top18bot12.d	95,000	99.1	97.6
top15bot15.d	105,000	99.1	97.5

The last column of Table 14 indicates that grouping based on class performance can improve the performance of the two-class network. The worst performing network was the one trained on file *1st2nd15.d*, in which the class groupings were essentially random. The best performance was by the network with the 18/12 split based on class performance ranking, closely followed by the even 15/15 split.

Each of the eight groups of classes were then used to train a network to determine the individual classes. Also, the bottom 15 classes of the ranked 15/15 split network were split again into the bottom 8 and bottom 7 classes and the 8- and 7-class networks were also trained and tested. This was done to determine if a further breakdown would result in any overall accuracy gain.

Experimental results: see Table 15

Data files: as indicated

Classes: as indicated

Training vectors: the lesser of 100 or half the total, per class

Test vectors: 8469 total (half of each class)

Hidden nodes: 16

Table 15. Second-Level Network Test Performance

File name	Classes	Iterations	% Correct
1st15.d	15	100,000	97.3
2nd15.d	15	100,000	99.0
top21.d	21	60,000	99.4
bot9.d	9	135,000	97.0
top18.d	18	70,000	99.7
bot12.d	12	125,000	96.1
top15.d	15	110,000	99.9
bot15.d	15	70,000	95.6
bot8and7.d	2	85,000	96.8
bot8.d	8	95,000	99.3
bot7.d	7	90,000	97.1

Iterations: as indicated

The end-to-end probability of the hierarchical systems is shown in Table 16, with the *1st2nd15* system added for comparison. The latter was outperformed by all 3 of the hierarchical systems. The three-tiered hierarchical system did not improve the overall performance of the first two tiers. The calculation of the overall probabilities is explained in the next section.

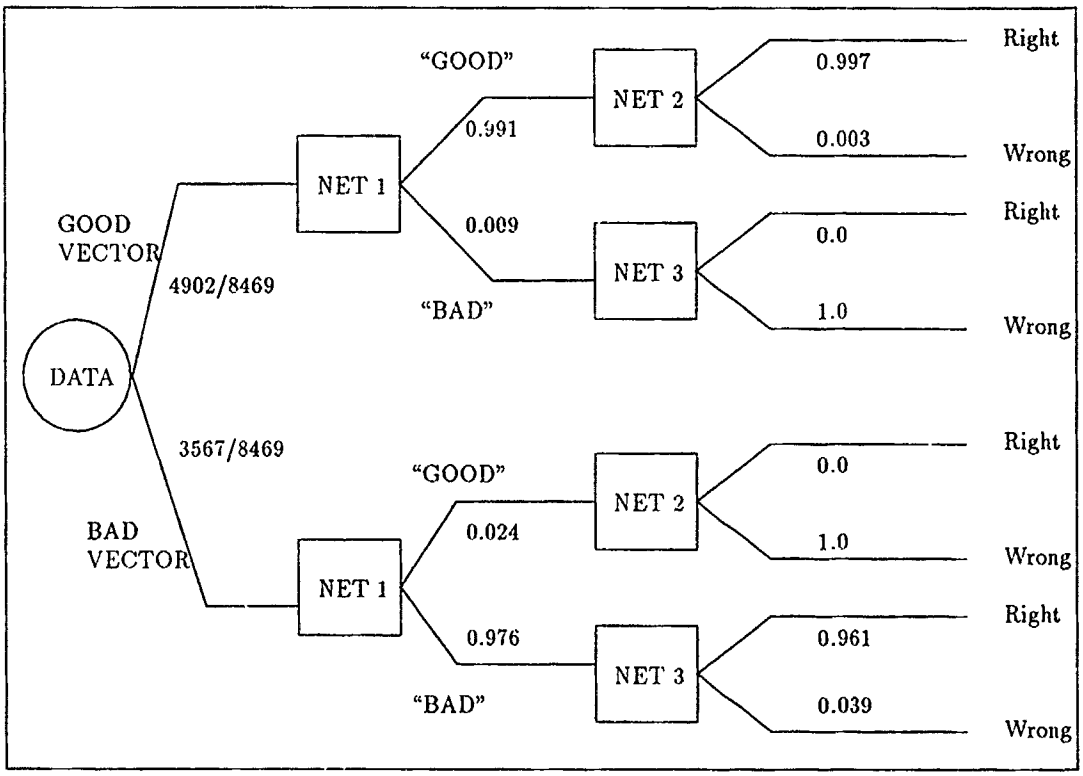
The best performance was delivered by the system which split the 30 classes 18/12 based on the class performances in the 30-class network. The accuracy of 96.5% is a 1.3% improvement over the equivalent 30-class performance of 95.2%.

Table 16. Overall Hierarchical System Performances

System name	Class Split	Classification %
1st2nd15	15/15	93.4
top21bot9	21/9	95.3
top18bot12	18/12	96.5
top15bot15	15/15	95.8

The performance of each network on each class of vector is included for reference in Table 35 (Appendix C). The "dash" entries indicate classes which would never be presented to that network.

Figure 8. Probability Tree for Hierarchical System



6.7 Probability Tree

The probability tree for the hierarchical system is shown in Figure 8. The probability of a vector from a particular class being presented to NET1 is equal to the fraction of the total number of test vectors that class represents. NET1 splits the 30 classes into two groups of classes, called "Good" and "Bad". The probability of an input vector being "Good" or "Bad" is equal to the fraction of the total test vectors represented by all the classes in that group. NET2 classifies the "Good" vectors and NET3 classifies the "Bad" vectors. Note that the sub-networks NET2 and NET3 score 0% correct if NET1 preceding it makes an error. This is because the sub-networks cannot correctly classify a vector from a class they have not been trained on.

The probabilities shown are for the best-performing system. The other systems could be represented by similar structures except that the three-level system would require an extra column to represent the 8/7 split of the bottom 15

classes. The probability of a correct classification is found by recursively multiplying and adding the probabilities of the left-most node's subtrees for the branches with "Right" outcomes. For the best system (top18bot12), the overall probability is $P = (4902/8469)(0.991)(0.997) + (3567/8469)(0.976)(0.961) = 0.965$. The overall probabilities for each of the networks is given in percentage form in Table 16.

6.8 Summary

The class groupings based on the percentage of input classes correctly classified showed a greater spread and a higher maximum than those based on the number of times an output class was incorrectly chosen. Therefore, the input classification accuracy was used to partition the 30 classes into groups for the multi-network systems.

The parallel neural networks did not provide an improvement in system classification accuracy. All three of the hierarchical systems provided an improvement in classification accuracy over the single 30-class network. The maximum accuracy achieved was 96.5% by the hierarchy based on a 18/12 class partition.

The following final chapter summarizes the findings of the thesis.

VII. Conclusions

7.1 Introduction

This chapter summarizes the findings of the experiments of chapters 4, 5, and 6. The factors affecting the classification accuracies of the neural networks and the methods identified for improving the accuracy are discussed. Recommendations for further research are included, and the most important findings are restated in the final summary.

7.2 Factors Affecting Network Accuracy

The factors affecting the classification accuracy of the neural networks in the experiments are listed in Table 17. The effect of increasing various network or data parameters is shown. Further clarification can be found in the applicable sections.

Table 17. Factors Affecting Network Accuracy

Section	Parameter	Increase Effect	Comments
4.2/4.3	Number of trng vectors	increase	minimum required
4.3/4.4	Number of classes	decrease	
4.3/4.5	Number of iterations	increase	maximum exists
4.6	Binary Coding	decrease	worse for many classes
4.8	Number of hidden nodes	increase	max = no. classes
4.9	Rel. no. trng vectors	variable	class-by-class effects
4.10	Equal-valued features	decrease	
4.11	Number of features	increase	even when correlated
4.12	Saliency of features	increase	low correlation
5.4	Feature correlation	decrease	
5.5	Total deviation	decrease	
5.7	Distinction	increase	

7.3 Methods of Improving Accuracy

The accuracy of the experimental neural networks were improved by controlling the parameters listed in Table 17. However, in a practical application, the designer of a neural network classification system does not have control over many of those

parameters. The number of hidden layer nodes can be optimized, as can the relative numbers of training vectors (subject to data availability). Binary coding of the output nodes should not be used. All of the available features should be used. The number of iterations of training can be controlled to achieve maximum accuracy. The rest of the parameters listed are not under the control of the designer.

Systems of neural networks can be used to increase the overall classification accuracy. In this case, the designer has control over how the problem is divided between the separate neural networks. Partitioning of the classes based on the class-by-class performance of a network using all of the classes proved to be the best method in this research.

The use of parallel neural networks, in which separate networks were trained to recognize a subset of the total number of classes, did not improve the experimental accuracy. The problem was that the neural networks responded unpredictably to vectors from classes they had not been trained with.

Hierarchical neural networks were found to provide an increase in the overall classification accuracy. The method was to partition the classes into two groups with one neural network, and then classify the vectors of each group with two other networks. The number of classes included in each group was determined empirically. The partition which produced the highest accuracy was the 18/12 split, which resulted in an overall accuracy of 96.5% (see Table 16). This was also the partition which gave the highest accuracy for grouping into two classes (see Table 14), but the margin of victory (0.1%) and the limited number of partitions tested (only 4) precludes any further generalization.

7.4 Recommendations

Many of the sections of this thesis could be explored further. The results of similar experiments on different neural network topologies and using different data could be informative. The use of different criteria for the partitioning of the classes for the hierarchical systems may yield further increases in classification accuracy.

7.5 Summary

This research has demonstrated that neural networks can be used to design radar classification systems with very high accuracy for 30 classes. It has also demonstrated that the hierarchical approach to the classification problem with 30 classes

results in greater overall system accuracy. The method used should be applicable to other problems with many classes. The maximum classification accuracy on a set of 8469 test vectors representing half of each class was 96.5%.

Appendix A. *Chapter 4 Data Tables*

This appendix contains several data tables which supplement the sections referenced in the table captions. Some of the data from each of these tables is graphed in the applicable section.

Table 18. Convergence of Network Accuracy (section 4.5)

File name	Iterations	% (Test/Train)
a50x20.d	10,000	91.0/91.9
"	20,000	90.8/94.5
"	30,000	92.6/94.5
"	40,000	91.8/94.5
"	50,000	91.8/96.1
"	60,000	92.4/96.6
"	70,000	92.0/97.1
"	80,000	94.0/97.0
"	90,000	92.8/96.7
"	100,000	92.8/96.7
"	110,000	92.6/96.3
"	120,000	93.0/96.9
"	150,000	93.2/97.4
"	200,000	91.6/97.5
a125x26.d	10,000	76.0/77.1
"	20,000	88.3/89.8
"	30,000	90.9/91.9
"	50,000	94.0/94.6
"	75,000	91.2/94.8
"	100,000	93.1/94.5
"	125,000	93.4/95.3
"	150,000	93.1/95.2

Table 19. Effect of Number of Hidden Nodes (section 4.8)

			Percentage Correct (Test/Train)		
		Hidden	Number of iterations		
File name	Classes	nodes	10,000	20,000	40,000
a125x08.d	8	75	98/98.5	98.5/99.4	98.5/99.8
"	8	50	98.0/98.3	98.5/99.5	98.5/99.9
"	8	25	98.0/98.9	98.5/99.6	98.5/99.9
"	8	16	98.5/98.5	99.0/99.3	98.5/99.5
"	8	8	97.5/97.8	99.0/98.8	98.5/99.5
"	8	5	97.0/94.8	96.5/96.1	95.0/96.3
"	8	3	77.5/75.4	72.5/74.3	90.5/91.5
"	8	2	45.0/47.0	63.0/62.2	crash!
a50x14.d	14	40	96.9/97.7	95.7/97.3	97.1/99.3
"	14	30	96.3/97.3	95.7/97.1	96.3/98.7
"	14	20	95.4/96.6	94.6/96.9	95.7/98.7
"	14	16	95.1/97.4	95.7/97.1	96.0/98.0
"	14	10	94.9/95.4	94.3/94.3	95.1/97.6
"	14	6	85.1/86.4	84.0/87.4	90.3/93.1
"	14	4	78.9/77.3	78.0/80.4	86.9/89.7
"	14	3	51.2/52.3	41.0/42.7	57.1/56.9
a100x20.d	20	40	88.2/89.3	91.4/93.6	94.6/96.7
"	20	20	87.8/89.5	92.4/94.2	94.0/96.2
"	20	12	86.0/86.7	91.6/93.1	92.4/94.4
"	20	8	83.6/86.7	87.2/89.9	89.2/92.3
"	20	5	64.8/66.1	64.8/66.5	69.6/71.0
a75x26.d	26	45	83.9/85.0	93.1/91.0	94.5/95.4
"	26	26	80.2/80.0	91.1/93.0	92.8/95.0
"	26	12	76.1/77.1	84.9/86.5	89.4/90.3
"	26	10	70.5/72.5	88.6/88.3	89.4/89.9
"	26	8	70.5/72.9	77.7/80.0	77.1/77.7
"	26	6	13.1/44.8	54.9/55.7	59.1/60.9

Table 20. Effect of Numbers of Training Vectors (section 4.9)

	Percentage Accuracy (Test Vectors) for file:				
Class	alldata.d	max100.d	all100.d	some200.d	some300.d
1	68.8	72.8	52.8	72.8	64.8
2	39.2	56.8	48.8	57.6	56.0
3	98.3	93.3	95.0	94.4	95.2
4	99.2	100	59.2	100	100
5	100	100	97.6	100	100
6	98.5	97.5	98.5	94.0	98.0
7	100	99.7	99.7	100	99.6
8	0.0	100	100	100	35.7
9	31.0	100	100	100	100
10	94.1	97.9	94.4	90.0	94.1
11	90.1	90.8	94.9	94.9	94.0
12	100	100	100	99.7	99.7
13	100	100	100	100	100
14	99.3	98.3	99.0	99.3	100
15	100	100	100	100	100
16	99.7	93.9	98.3	97.6	99.3
17	97.0	99.7	97.0	98.7	96.6
18	97.2	95.7	99.6	98.2	89.0
19	66.2	59.5	21.6	71.6	87.2
20	99.7	100	97.6	99.3	99.3
21	100	100	100	100	100
22	100	100	100	100	100
23	97.6	97.3	89.9	94.6	91.3
24	99.3	99.6	97.5	98.9	97.1
25	97.4	97.1	98.3	95.7	93.0
26	64.6	100	100	95.8	95.8
27	100	100	100	100	100
28	81.7	88.6	81.2	90.1	81.2
29	98.0	89.9	85.9	96.6	93.3
30	100	92.5	69.6	95.2	93.0
Overall	93.7	95.2	91.3	95.8	95.0

Appendix B. *Chapter 5 Data Tables*

This appendix contains large data tables giving statistics for every class and feature. Discussion of the data can be found in the sections referenced in the table captions.

Table 21. Feature Means by Class (1 of 2)(section 5.3)

	Feature Number							
Class	f01	f02	f03	f04	f05	f06	f07	f08
1	90	90	100	119	138	153	159	155
2	97	96	100	107	114	116	116	114
3	130	145	160	166	162	147	127	109
4	130	124	117	111	109	114	123	133
5	128	124	119	116	114	117	123	129
6	83	70	60	57	64	79	98	116
7	171	185	191	189	180	169	161	156
8	128	129	129	129	129	129	129	129
9	104	102	106	113	121	127	130	129
10	190	200	197	180	158	137	124	119
11	161	170	174	171	164	155	146	140
12	183	205	220	227	224	216	204	192
13	102	104	110	116	121	122	122	122
14	207	230	239	234	216	193	173	159
15	180	196	205	206	201	196	195	198
16	108	115	127	141	152	158	157	152
17	92	89	94	105	118	129	136	138
18	129	149	170	185	192	189	182	175
19	125	128	131	134	136	136	136	134
20	106	109	116	125	132	136	136	135
21	205	229	243	244	234	218	205	197
22	170	181	186	183	176	168	163	160
23	142	152	164	173	177	176	170	162
24	139	136	131	126	124	126	131	137
25	112	144	144	141	139	136	134	133
26	129	123	116	110	110	116	124	133
27	121	124	127	129	129	127	123	120
28	74	57	48	51	63	80	96	109
29	100	105	116	129	140	116	147	144
30	110	107	106	110	115	121	126	130
31	77	68	70	82	100	119	131	135
32	97	93	94	99	106	113	119	122

Table 22. Feature Means by Class (2 of 2)(section 5.3)

	Feature Number							
Class	f09	f10	f11	f12	f13	f14	f15	f16
1	145	134	125	124	125	128	130	132
2	113	114	117	122	124	125	124	121
3	98	96	96	96	96	96	96	96
4	140	142	138	131	125	121	121	123
5	133	134	132	128	125	123	124	126
6	131	142	146	149	149	148	147	146
7	153	151	150	147	150	150	145	130
8	129	129	129	129	129	129	129	129
9	128	126	125	125	126	126	125	125
10	120	122	124	123	122	122	124	126
11	137	137	139	140	141	143	144	145
12	182	172	164	155	147	140	134	130
13	123	127	132	138	144	147	148	150
14	152	148	147	141	134	127	119	113
15	204	207	207	197	183	167	152	140
16	144	138	132	130	127	125	123	123
17	137	136	135	137	138	140	140	140
18	171	167	162	155	143	128	117	112
19	132	131	130	130	129	129	129	129
20	133	133	133	136	137	137	137	136
21	196	199	205	207	205	202	195	189
22	160	161	162	160	158	156	154	154
23	154	148	146	146	147	149	152	154
24	140	142	141	140	140	143	148	152
25	132	129	127	124	121	119	116	112
26	138	137	132	126	121	119	120	123
27	119	119	120	122	122	121	120	120
28	116	119	119	120	121	122	121	126
29	140	137	136	136	136	135	134	134
30	132	132	130	128	127	125	124	124
31	133	129	124	123	124	125	125	124
32	124	125	125	127	128	128	130	132

Table 23. Feature Deviation by Class (1 of 2)(section 5.3)

	Feature Number							
Class	f01	f02	f03	f04	f05	f06	f07	f08
1	19	19	18	17	18	18	18	16
2	23	27	37	48	54	52	43	29
3	10	9	10	15	19	18	12	8
4	2	2	2	1	1	1	2	2
5	1	1	1	1	1	1	1	1
6	24	38	48	53	54	51	45	39
7	5	5	4	3	4	4	4	4
8	1	1	1	1	1	1	1	1
9	2	2	2	2	1	1	1	1
10	16	16	13	10	8	7	6	5
11	11	11	10	8	6	6	6	6
12	6	6	6	5	4	4	4	4
13	13	14	11	8	5	6	6	6
14	13	13	11	11	11	10	9	8
15	13	12	11	10	9	8	7	7
16	14	13	10	6	4	4	5	5
17	10	11	10	7	4	2	2	2
18	39	35	26	16	17	19	18	14
19	14	13	10	6	5	6	6	6
20	6	6	4	3	2	2	2	2
21	11	11	8	6	6	5	5	4
22	6	6	5	3	3	3	2	2
23	11	10	8	6	7	7	8	7
24	12	12	10	8	7	5	4	4
25	13	13	11	7	5	5	6	5
26	2	2	2	2	1	2	2	2
27	3	3	2	2	2	2	2	2
28	8	11	14	15	16	15	14	13
29	13	13	10	6	3	3	1	3
30	3	4	3	2	2	2	2	2
31	14	17	16	13	8	4	5	5
32	10	10	8	6	4	4	4	4

Table 24. Feature Deviation by Class (2 of 2)(section 5.3)

	Feature Number							
Class	f09	f10	f11	f12	f13	f14	f15	f16
1	18	20	20	19	20	19	18	17
2	21	21	22	22	22	22	22	21
3	13	16	16	16	16	16	16	16
4	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1
6	35	31	31	30	32	34	36	37
7	4	4	4	4	3	4	4	7
8	1	1	1	1	1	1	1	1
9	1	0	1	1	1	1	1	1
10	5	5	5	5	5	5	5	5
11	6	6	6	6	6	7	7	7
12	4	4	4	3	3	3	3	3
13	6	6	6	5	5	5	5	4
14	7	6	6	5	6	5	5	4
15	6	6	6	6	6	6	6	5
16	4	3	3	3	3	3	2	2
17	2	1	1	1	1	1	1	1
18	10	9	9	11	11	10	8	7
19	5	4	3	3	3	3	3	3
20	1	1	1	1	1	1	1	1
21	4	4	5	4	5	5	5	5
22	2	2	2	2	2	2	2	2
23	6	5	6	6	7	7	7	7
24	3	4	4	3	3	3	3	3
25	4	3	3	3	4	4	4	3
26	1	1	1	1	1	1	1	1
27	1	1	1	1	1	1	1	1
28	11	11	11	11	11	11	10	11
29	3	2	2	2	3	3	2	2
30	2	2	2	2	2	2	2	2
31	1	3	3	3	3	3	3	4
32	4	1	1	4	4	4	1	4

Table 25. Class Ranking by Total Feature Deviation (section 5.5)

Rank	Class	Deviation
1	5	2.9
2	8	4.3
3	9	4.9
4	4	5.5
5	26	6.0
6	27	6.5
7	20	10.7
8	22	12.8
9	12	17.1
10	7	17.3
11	17	19.9
12	30	22.1
13	29	23.4
14	21	24.8
15	16	25.1
16	24	25.3
17	25	27.1
18	19	27.4
19	11	29.3
20	23	29.6
21	13	30.0
22	15	32.5
23	10	34.2
24	14	34.4
25	28	48.9
26	3	58.2
27	1	73.8
28	18	74.3
29	2	130.1
30	6	158.7

Table 26. Distinction of Classes (1 of 4)(section 5.7)

	Class							
Class	c01	c02	c03	c04	c05	c06	c07	c08
1	0	5	19	14	14	9	23	13
2	5	0	11	8	7	6	22	8
3	19	11	0	32	31	17	37	27
4	14	8	32	0	28	12	108	61
5	14	7	31	28	0	12	124	52
6	9	6	17	12	12	0	18	13
7	23	22	37	108	124	18	0	101
8	13	8	27	61	52	13	101	0
9	9	4	33	66	58	11	121	51
10	19	11	18	41	37	17	36	32
11	16	15	23	48	51	14	18	40
12	37	30	55	154	174	25	44	150
13	11	7	27	32	28	7	50	26
14	26	22	37	65	72	22	26	67
15	37	33	53	115	125	26	42	113
16	6	9	26	39	40	13	42	34
17	9	8	37	55	63	7	65	50
18	15	16	23	34	38	15	16	35
19	8	8	22	26	20	11	47	6
20	9	9	33	66	66	10	76	41
21	54	45	74	197	219	34	84	196
22	27	26	44	153	183	19	17	147
23	18	18	28	60	66	14	14	55
24	13	12	29	34	41	9	41	31
25	12	9	17	31	29	14	51	21
26	14	7	31	13	25	12	108	55
27	14	5	23	61	61	14	109	44
28	16	6	29	31	31	5	61	38
29	6	9	30	44	49	10	47	34
30	10	5	32	29	21	10	99	38
31	10	5	30	26	22	7	64	27
32	10	3	29	29	26	8	75	29

Table 27. Distinction of Classes (2 of 4)(section 5.7)

	Class							
Class	c09	c10	c11	c12	c13	c14	c15	c16
1	9	19	16	37	11	26	37	6
2	4	11	15	30	7	22	33	9
3	33	18	23	55	27	37	53	26
4	66	41	48	154	32	65	115	39
5	58	37	51	174	28	72	125	40
6	11	17	14	25	7	22	26	13
7	121	36	18	44	50	26	42	42
8	51	32	40	150	26	67	113	34
9	0	34	53	174	24	75	129	42
10	34	0	22	61	30	31	63	28
11	53	22	0	50	26	30	47	23
12	174	61	50	0	82	27	32	76
13	24	30	26	82	0	53	70	33
14	75	31	30	27	53	0	43	35
15	129	63	47	32	70	43	0	74
16	42	28	23	76	33	35	74	0
17	63	44	28	108	16	59	94	41
18	41	26	19	17	33	14	26	22
19	22	21	22	76	20	41	74	16
20	63	40	30	123	19	62	100	35
21	221	99	80	55	108	60	28	122
22	186	56	27	57	60	42	44	66
23	70	35	14	41	33	27	39	30
24	47	36	17	77	13	46	69	33
25	28	18	25	83	29	39	80	21
26	56	37	48	151	33	67	118	37
27	54	24	51	158	32	69	123	42
28	27	30	42	77	24	53	72	32
29	48	35	20	83	22	45	79	19
30	19	35	46	142	21	66	112	32
31	16	27	37	94	24	51	89	28
32	19	29	41	107	16	59	94	35

Table 28. Distinction of Classes (3 of 4)(section 5.7)

	Class							
Class	c17	c18	c19	c20	c21	c22	c23	c24
1	9	15	8	9	54	27	18	13
2	8	16	8	9	45	26	18	12
3	37	23	22	33	74	44	28	29
4	55	34	26	66	197	153	60	34
5	63	38	20	66	219	183	66	41
6	7	15	11	10	34	19	14	9
7	65	16	47	76	84	17	14	41
8	50	35	6	41	196	147	55	31
9	63	41	22	63	221	186	70	47
10	44	26	21	40	99	56	35	36
11	28	19	22	30	80	27	14	17
12	108	17	76	123	55	57	41	77
13	16	33	20	19	108	60	33	13
14	59	14	41	62	60	42	27	46
15	94	26	74	100	28	44	39	69
16	41	22	16	35	122	66	30	33
17	0	36	24	19	157	98	43	19
18	36	0	26	35	45	19	14	29
19	24	26	0	15	120	69	34	20
20	19	35	15	0	173	114	46	22
21	157	45	120	173	0	89	68	111
22	98	19	69	114	89	0	16	54
23	43	14	34	46	68	16	0	25
24	19	29	20	22	111	54	25	0
25	39	25	12	30	125	76	37	30
26	61	36	25	67	201	159	62	11
27	81	39	24	74	205	166	65	51
28	28	34	27	31	99	70	17	35
29	20	27	15	15	131	72	32	21
30	37	36	17	38	185	138	60	36
31	28	34	19	32	135	90	19	33
32	27	37	19	33	145	99	53	33

Table 29. Distinction of Classes (4 of 4)(section 5.7)

	Class							
Class	c25	c26	c27	c28	c29	c30	c31	c32
1	12	14	14	16	6	10	10	10
2	9	7	5	6	9	5	5	3
3	17	31	23	29	30	32	30	29
4	31	13	61	34	44	29	26	29
5	29	25	61	34	49	21	22	26
6	14	12	14	5	10	10	7	8
7	51	108	109	61	47	99	64	75
8	21	55	44	38	34	38	27	29
9	28	56	54	27	48	19	16	19
10	18	37	24	30	35	35	27	29
11	25	48	51	42	20	46	37	41
12	83	154	158	77	83	142	94	107
13	29	33	32	24	22	21	24	16
14	39	67	69	53	45	66	51	59
15	80	118	123	72	79	112	89	94
16	21	37	42	32	19	32	28	35
17	39	61	84	28	20	37	28	27
18	25	36	39	34	27	36	34	37
19	12	25	24	27	15	17	19	19
20	30	67	74	34	15	38	32	33
21	125	201	205	99	131	185	135	145
22	76	159	166	70	72	138	90	99
23	37	62	65	47	32	60	49	53
24	30	41	51	35	21	36	33	33
25	0	25	22	29	28	25	21	28
26	25	0	51	32	46	25	22	28
27	22	51	0	29	55	45	29	30
28	29	32	29	0	32	27	12	17
29	28	46	55	32	0	35	31	31
30	25	25	45	27	35	0	15	17
31	21	22	29	12	31	15	0	13
32	28	28	30	17	31	17	13	0

Appendix C. *Chapter 6 Data Tables*

This appendix contains tables of the classification performance of various networks on a class-by-class basis. Discussions of the table data can be found in the sections referenced in the table captions.

Table 30. Classification Confusion Matrix (1 of 3)(section 6.2)

Output	Input Class									
Class	c01	c02	c03	c04	c05	c06	c07	c08	c09	c10
1	91	4	0	0	0	0	0	0	0	0
2	5	71	0	0	0	5	0	0	0	1
3	0	0	1018	0	0	0	0	0	0	0
4	2	0	0	125	0	0	0	0	0	0
5	0	1	0	0	125	0	0	0	0	0
6	0	4	0	0	0	195	0	0	0	0
7	0	0	0	0	0	0	1089	0	0	0
8	0	12	21	0	0	0	0	42	0	0
9	0	2	0	0	0	0	0	0	42	0
10	0	2	14	0	0	0	0	0	0	332
11	1	0	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0
13	1	4	1	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0
16	4	0	0	0	0	0	0	0	0	0
17	2	0	1	0	0	0	0	0	0	0
18	4	0	0	0	0	0	0	0	0	0
19	1	2	8	0	0	0	0	0	0	2
20	1	6	1	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0
23	1	1	0	0	0	0	3	0	0	0
24	5	1	4	0	0	0	0	0	0	0
25	0	1	21	0	0	0	0	0	0	2
26	1	1	0	0	0	0	0	0	0	0
27	0	7	0	0	0	0	0	0	0	1
28	0	3	0	0	0	0	0	0	0	0
29	6	1	0	0	0	0	0	0	0	0
30	0	2	2	0	0	0	0	0	0	0
Totals	125	125	1091	125	125	200	1092	42	42	339
Errors	34	54	73	0	0	5	3	0	0	7
Not top 3	24	23	50	0	0	3	0	0	0	4

Table 31. Classification Confusion Matrix (2 of 3)(section 6.2)

Output	Input Class									
Class	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20
1	0	0	0	0	0	4	0	0	1	0
2	0	0	0	0	0	0	0	1	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	19	0
9	0	0	0	0	0	0	0	0	1	0
10	13	0	0	0	0	0	0	0	11	0
11	395	0	0	0	0	0	0	0	0	0
12	0	283	0	0	0	0	0	6	0	0
13	0	0	297	0	0	0	0	0	0	0
14	0	0	0	281	0	0	0	2	0	0
15	0	0	0	0	295	0	0	0	0	0
16	0	0	0	0	0	279	0	0	4	0
17	0	0	0	0	0	0	296	0	1	0
18	0	0	0	5	0	0	0	269	0	0
19	0	0	0	0	0	9	0	0	88	0
20	0	0	0	0	0	0	0	0	2	296
21	0	0	0	0	0	0	0	0	0	0
22	5	0	0	0	0	0	0	1	0	0
23	17	0	0	0	0	0	0	0	0	0
24	5	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	5	0	0	5	0
26	0	0	0	0	0	0	0	2	1	0
27	0	0	0	0	0	0	0	0	1	0
28	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	1	0	11	0
30	0	0	0	0	0	0	0	0	0	0
Totals	435	283	297	286	295	297	297	281	148	296
Errors	40	0	0	5	0	18	1	12	60	0
Not top 3	5	0	0	0	0	1	0	5	19	0

Table 32. Classification Confusion Matrix (3 of 3)(section 6.2)

Output	Input Class										Row
Class	c21	c22	c23	c24	c25	c26	c27	c28	c29	c30	Errors
1	0	0	0	0	0	0	0	0	0	0	9
2	0	0	0	0	0	0	0	4	0	0	16
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	2
5	0	0	0	0	0	0	0	0	0	4	5
6	0	0	0	0	0	0	0	7	0	0	11
7	0	0	2	0	0	0	0	0	0	0	2
8	0	0	0	0	0	0	0	0	0	4	56
9	0	0	0	0	0	0	0	0	0	4	7
10	0	0	0	1	1	0	0	0	0	0	42
11	0	0	4	0	0	0	0	0	1	0	7
12	0	0	0	0	0	0	0	0	0	0	6
13	0	0	1	0	0	0	0	1	0	1	9
14	0	0	0	0	0	0	0	0	0	0	2
15	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	1	0	0	0	0	0	9
17	0	0	0	0	0	0	0	2	0	0	6
18	0	0	0	0	0	0	0	0	0	0	9
19	0	0	0	0	1	0	0	0	23	0	16
20	0	0	0	0	0	0	0	0	6	0	16
21	261	0	0	0	0	0	0	0	0	0	0
22	0	292	1	0	0	0	0	0	0	0	7
23	0	0	289	0	0	0	0	0	0	0	22
24	0	0	0	274	0	0	0	8	0	0	23
25	0	0	0	0	112	0	0	0	0	0	31
26	0	0	0	0	0	18	0	1	0	0	6
27	0	0	0	0	0	0	63	0	0	0	9
28	0	0	0	0	0	0	0	179	0	17	20
29	0	0	0	0	0	0	0	0	267	0	22
30	0	0	0	0	0	0	0	0	0	368	1
Totals	261	292	297	275	115	18	63	202	297	398	407
Errors	0	0	8	1	3	0	0	23	30	30	407
Not top 3	0	0	2	1	0	0	0	16	4	3	160

Table 33. Classification Accuracy Ranking by Class (section 6.3)

Rank	Class	% Correct
1	4	100
2	5	100
3	8	100
4	9	100
5	12	100
6	13	100
7	15	100
8	20	100
9	21	100
10	22	100
11	26	100
12	27	100
13	7	99.7
14	17	99.7
15	24	99.6
16	14	98.3
17	10	97.9
18	6	97.5
19	25	97.4
20	23	97.3
21	18	95.7
22	16	93.9
23	3	93.3
24	30	92.5
25	11	90.8
26	29	89.9
27	28	88.6
28	1	72.8
29	19	59.5
30	2	56.8
Overall		95.2

Table 34. Confusion Frequency Ranking by Class (section 6.3)

Rank	Class	Errors
1	3	0
2	15	0
3	21	0
4	4	2
5	7	2
6	14	2
7	30	4
8	5	[illegible]
9	12	6
10	17	6
11	26	6
12	9	7
13	11	7
14	22	7
15	1	9
16	13	9
17	16	9
18	18	9
19	27	9
20	6	11
21	2	16
22	20	16
23	28	20
24	23	22
25	29	22
26	24	23
27	25	34
28	10	42
29	19	46
30	8	56
Total		407

Table 35. Class Identification by Networks (section 6.6)

	Network Training File Name (all have .d extension)								
Class	max100	1st15	2nd15	top21	bot9	top18	bot12	top15	bot15
1	72.8	87.9	-	-	81.4	-	82.0	-	75.7
2	56.8	61.6	-	-	69.3	-	80.4	-	78.0
3	93.3	91.5	-	-	93.2	-	95.2	-	96.0
4	100	93.9	-	95.7	-	99.1	-	99.1	-
5	100	93.9	-	98.1	-	99.1	-	99.1	-
6	97.5	92.0	-	97.1	-	98.1	-	-	95.6
7	99.7	93.9	-	97.2	-	99.1	-	99.1	-
8	100	93.9	-	98.1	-	99.1	-	99.1	-
9	100	93.9	-	98.1	-	99.1	-	99.1	-
10	97.9	89.2	-	96.4	-	97.6	-	-	93.5
11	90.8	92.0	-	-	94.0	-	90.6	-	93.7
12	100	93.9	-	97.8	-	98.8	-	99.1	-
13	100	93.3	-	98.1	-	99.1	-	99.1	-
14	98.3	93.9	-	98.1	-	99.1	-	-	96.8
15	100	93.9	-	98.1	-	99.1	-	99.1	-
16	93.9	-	93.2	-	93.8	-	96.0	-	96.5
17	99.7	-	96.8	97.1	-	98.4	-	99.1	-
18	95.7	-	96.4	95.3	-	-	97.2	-	95.1
19	59.5	-	92.5	-	83.4	-	81.1	-	79.7
20	100	-	97.1	97.8	-	98.8	-	98.4	-
21	100	-	97.1	98.1	-	99.1	-	99.1	-
22	100	-	97.1	97.8	-	99.1	-	99.1	-
23	97.3	-	96.1	96.5	-	-	95.0	-	92.2
24	99.6	-	96.8	97.4	-	98.4	-	98.7	-
25	97.4	-	96.3	97.2	-	-	96.8	-	93.3
26	100	-	97.1	98.1	-	97.0	-	99.1	-
27	100	-	97.1	98.1	-	99.1	-	99.1	-
28	88.6	-	96.6	-	93.7	-	95.7	-	92.2
29	89.9	-	91.5	-	90.4	-	95.0	-	94.9
30	92.5	-	97.1	-	94.0	-	97.6	-	92.6
Overall	95.2	91.4	96.1	97.5	91.4	98.8	93.8	99.0	93.2

Appendix D. *Software*

This appendix includes the listings of many of the C programs and Unix script files written to manipulate data vectors, analyze data, and produce tables. Not all programs have been included.

D.1 newdata

```
% newdata
% This script file uses other script files to pick specific lines of
% each of the 30 data files.  Since the mixall script pre-mixes the
% vectors of each file, a different and pseudo-random set of vectors is
% chosen each time this script is used.
% The output data file is ready for running on Neural Graphics.
%
cat  newheader  > trgvect50
cat  newheader  > testvect25
mixall
pickmed c01 c02
pickvlg c03
pickmed c04 c05
picklg c06
pickvlg c07
picksm c08 c09
picklg c1[0-8]
pickmed c19
picklg c2[0-4]
pickmed c25
picksm c26 c27
picklg c28 c29 c30
for file in trgvect50 testvect25
do
ex - $file << endscript
1,1d
wq
endscript
done
cat -b trgvect50 testvect25 > tempz
```

```
cat newheader tempz > new50x30.d
rm tempz
echo 'New data file is called new50x30.d'
```

D.2 pickclass

```
% pickclass
% This script takes all command line arguments as class numbers
% and puts together a data file of those classes in the format
% required by Neural Graphics.  The script reads 60 files, one test
% and one training file for each of 30 classes.
%
echo 'Header' >new.trg
echo 'Header' >new.test
for num in $*
do
   cat c$num.trg >> new.trg
   cat c$num.test >> new.test
done
ex - new.trg << endscript
1,1d
wq
endscript
ex - new.test << endscr
1,1d
wq
endscr
wc -l new.trg >head1
awk '{print $1 }' head1 >head3
wc -l new.test >head2
awk '{print $1 }' head2 >head4
cat -b new.trg new.test >new.1
cat head3 head4 new.1 >new.2
reclass4 < new.2 > new.3
awk 'length < 20' new.3 >head5
awk 'length > 20' new.3 >new.4
cat head5 new.4 >new
rm new.trg
rm new.test
rm head?
```

```
rm new.?
echo 'New data file is called:  new'
```

D.3 renumber.c

```
/*********************************************************/
/* renumber.c
   Program to renumber a data file in Neural Network format.
   Does not print the line number and prints a new class number.
   First line of input file must be the number of lines and
   the new class number desired.   Std I/O.
  */

#include <stdio.h>
main()
{
    int count,lines, newclass;
    int a, b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r; /* 18 elms per vector */

scanf("%d %d",&lines, &newclass);

  for (count = 1; count <= lines  ; count++ )
   {
     scanf(" %d %d %d  %d %d %d %d %d %d %d %d %d  %d %d %d %d %d %d \n",
         &a, &b,&c,&d,&e,&f,&g,&h,&i, &j,&k,&l,&m,&n,&o,&p,&q,&r);


/* Don't print the line number , print new class number */
     printf("%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",
            b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,newclass);

    }
} /* end of main */
```

D.4 reclass.c

```
/*********************************************************/
/* reclass.c
```

```
  Program to convert original class number to get sequential as
 required by Greg Tarr's neural net modelling program.
Input file must start with number of vectors,
  oldclass, newclass line, then vectors.
  The lines must not be numbered yet.  */

#define MAXLINE 1000  /* longest line allowed */
#include <stdio.h>
main()
{
    char line1[MAXLINE] ;
    int linelen, numvect, oldclass, newclass, index;
    scanf("%d %d %d  \n", &numvect,&oldclass, &newclass);
   /*  printf("%d testvects 16 %d \n", numvect,newclass); */

    while ((linelen= (getline(line1,MAXLINE))) != 0)
       {
        line1[linelen -1] = '\0';  /* remove the /n */
if (linelen > 9)  {
  /* first remove the old class number at end of each line */
          index = linelen - 2;
          while (line1[index--] == ' ') ;   /* skip spaces */
          line1[index+1] = '\0';
          while (line1[index--] != ' ')  {
            line1[index+1] = '\0';          }

          printf("%s %d  \n",  line1, newclass);
 }
      }  /* only prints long data lines */
   /*
   printf("The previous data class was %d \n", oldclass);
   */
}

getline(s,lim)   /* get line into s, return length */
char s[];
int lim;
{
   int c,i;
   for (i=0; i<lim-1 && (c=getchar())!= EOF && c!='\n';++i)
      s[i] = c;
   if (c == '\n') {
```

```
      s[i] = c;
      ++i;
   }
   s[i] = '\0';
   return(i);
}
```

D.5 getconst.c

```
/*****************************************************/
/* getconst.c
   program to extract from a data file already in the format
 required by Greg Tarr's neural net modeling program to
print  all constant-valued vectors (all features equal).
Input and output are std and must be redirected by shell.
  */

#include <stdio.h>
main()
{
    int numvect, testvect, features, output, count;
    int a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r; /* 18 elms per vector */
    scanf("%d %d %d %d \n", &numvect,&testvect, &features, &output);

  for (count = 1; count <= (numvect + testvect); count++)
   {
     scanf("%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",
          &a,&b,&c,&d,&e,&f,&g,&h,&i, &j,&k,&l,&m,&n,&o,&p,&q,&r);

     if ((b==c && c==d && d==e && e==f) && (f==g && g==h && h==i && i==j)
      && (j==k && k==l && l==m && m==n) && (n==o && o==p && p==q))


     printf("%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",
           a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r);

    }

}
```

D.6 splitdata.c

```
/*******************************************************/
/* splitdata.c
program to Split a data file by writing the first and every
other line to the output.
Use to split exemplars into training and test groups.
Std I/O.
Must change the for loop counter to match size of file.
  */

#include <stdio.h>
main()
{
    int count;
    int a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q; /* 17 elms per vector */

  for (count = 1; count <= 16939 ; count++ )
   {
     scanf("%d %d  %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",
          &a,&b,&c,&d,&e,&f,&g,&h,&i, &j,&k,&l,&m,&n,&o,&p,&q);

     if ( count % 2 == 0)
     printf("%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d  \n",
           a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q);

    }

}
```

D.7 distinction.c

```
/*******************************************************/
/* distinction.c
   program to calculate the distinctiveness of classes of data
```

```
   based on their feature means and std devs.
   This is treating the cluster center of each class
   as the mean of its features.
   4 tables showing the "distance" between each pair of
   classes is produced.
   Reads the means and std devs from the file "allstats.d".
   Std input and output redirection to be used.
  */

#include <stdio.h>
#include <math.h>
#define ABS(X)   ((X) > -(X) ? (X) : -(X))
#define SQR(x)   ((x)*(x))

main()
{
    int x,y,z;
    double means[33][17], devs[33][17];
    double dist[33][33], sumdist;


/* read in the means for each class, feature */

   for (x=1; x<=32; x++)
      for (y=1; y<=16; y++)
 scanf("%lf",&means[x][y]);

/* read in the stddevs for each class, feature */

   for (x=1; x<=32; x++)
      for (y=1; y<=16; y++)
 scanf("%lf",&devs[x][y]);

/*  compute the matrix of distances between centers */

   for (x=1; x<=32; x++)
      for (y=1; y<=32; y++)
       {
 if (x==y) dist[x][y] = 0;
 else
  {
    sumdist = 0.0;
```

```
    for (z=1; z<=16; z++)
              sumdist += ABS(means[x][z]-means[y][z])/(devs[x][z]+devs[y][z]);
           dist[x][y] = sumdist;
         }
       }

/* now print the distances in four tables, 8x32 each */

  for (x = 1; x <= 32 ; x++)
    {
      printf("%4d",x);
      for (y=1; y<=8 ; y++)
printf("%5.0f %c", dist[x][y], ' ');
      printf("\n");
     }
   printf("\n");
  for (x = 1; x <= 32; x++)
    {
      printf("%4d",x);
      for (y=9; y<=16; y++)
printf("%5.0f %c", dist[x][y], ' ');
      printf("\n");
     }
   printf("\n");
  for (x = 1; x <= 32; x++)
    {
      printf("%4d",x);
      for (y=17; y<=24; y++)
printf("%5.0f %c", dist[x][y], ' ');
      printf("\n");
     }
   printf("\n");
  for (x = 1; x <= 32; x++)
    {
      printf("%4d",x);
      for (y=25; y<=32; y++)
printf("%5.0f %c", dist[x][y], ' ');
      printf("\n");
     }
   printf("\n");

}  /* end of main */
```

```
double sqrt(num)
/* returns the square root using Newton's method */
double num;
{
 double guess;
 guess = 100.0;
do
   guess = guess- (guess*guess-num)/(2.0*guess);
while (ABS(guess*guess-num) > 0.00000001);
return(guess);
}
```

D.8 feature.c

```
/*****************************************************/
/* feature.c
   Program to select only some of the features of each vector.
   Can be modified to select only desired features by
   changing the printf statement.
  */

#include <stdio.h>
main()
{
    int x,y, classes, trgvect, testvect, trgnum , testnum ;
    int a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r; /* 18 elms per vector */
    scanf("%d %d %d %d \n", &trgvect, &testvect,&trgnum, &testnum);
    classes = trgvect/trgnum;
    printf("%d %d %d %d \n", trgvect,testvect, trgnum, testnum);

  for (x=1; x <= testvect+trgvect; x++)  /* for every vector */
   {
     scanf("%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",
          &a,&b,&c,&d,&e,&f,&g,&h,&i, &j,&k,&l,&m,&n,&o,&p,&q,&r);

    /* print only desired features */
     printf(" %d %d %d %d %d   %d %d \n",
           a,b,c,l,m,n,r);  /* class is x */
```

```
    }
} /* end of main */
```

D.9 stat.c

```
/*******************************************************/
/* stat.c
   Program to calculate the mean and dev
   of the 16 features of each of 32 classes in a file.
   Also calculates the mean, dev of the features, independently
   of the class number.
   File must be in format required by Greg Tarr's program.
   Std input and output redirection to be used.
  */

#include <stdio.h>
#include <math.h>
#define TOTAL 17002  /* total number of vectors */
#define ABS(X)   ((X) > -(X) ? (X) : -(X))
#define SQR(x)   ((x)*(x))
main()
{
    int trgvect, testvect, insize, outsize, feature, class, vect[18];
    int count, count2, x  ;
    double sums[33][33];
    /* sums[class][n,sum(x),sum(x*x)] */
    double hold, dev, feat[33];
    scanf("%d %d %d %d \n", &trgvect, &testvect, &insize, &outsize);

/* first initialize sum array to all 0's */
    for (count =0; count <= 32; count ++)
        for (count2 =0; count2<= 32; count2 ++)
   sums[count][count2] = 0.0;

    for (count = 0; count <= 32; count++)
       feat[count] = 0.0;

/* read each vector in the file and update sums */
```

```
 for (count = 1; count <= (trgvect + testvect); count++)
  {
    for (count2 = 0; count2 <= 17; count2 ++)
      scanf("%d", &vect[count2]);
    x = vect[17];   /* the class number */
    sums[x][0] += 1;

    for (feature = 1; feature <= 16; feature++ )
{
 sums[x][feature] += vect[feature];
 sums[x][16+feature] += vect[feature]*vect[feature];
 feat[feature] += vect[feature];
 feat[16+feature] += vect[feature]*vect[feature];
       }
   }
/* now compute the means and std dev for each feature, class */

 for (x = 1; x <= 32; x++)
   {
     printf("\n");
     printf("class %d vectors %.0f \n", x, sums[x][0]);
     printf("means and std devs for 16 features: \n");
     for (feature = 1; feature <= 16; feature++)
  {
   printf("%.1f %c ",sums[x][feature]/sums[x][0], ' ' );
   if (feature==8 || feature==16)  printf("\n");
   }
     for (feature = 1; feature <= 16; feature++)
 {
     hold=sums[x][16+feature]-SQR(sums[x][feature])/sums[x][0];
          dev = sqrt(hold/(sums[x][0] -1));
          printf("%.1f %c ", dev, ' ' );
          if (feature==8 || feature==16)  printf("\n");
        }
    }
/* now compute the means and stddev for each feature.  */

     printf(" \n");
     for (feature = 1; feature <= 16; feature++)
   {
          hold = feat[16+feature]- SQR(feat[feature])/TOTAL;
```

```
    dev = sqrt(hold/(TOTAL - 1));
    printf("feature %d mean %.1f std dev %.1f \n", feature,
    feat[feature]/TOTAL, dev);
    }

} /* end of main */

double sqrt(num)
/* returns the square root using Newton's method */
double num;
{
 double guess;
 guess = 100.0;
do
   guess = guess- (guess*guess-num)/(2.0*guess);
while (ABS(guess*guess-num) > 0.00000001);
return(guess);
}
```

D.10 totdev.c

```
/*********************************************************/
/* totdev.c
   Program to calculate the total deviation from
   the cluster centers of each class of data.
   The center coordinates are defined as the mean of
   each feature for the class.  The total deviation is the sqrt
   of the sum of the variances of each feature.
   Reads the std devs from the file "alldevs".
   Std input and output redirection to be used.
  */

#include <stdio.h>
#include <math.h>
#define ABS(X)   ((X) > -(X) ? (X) : -(X))
#define SQR(x)   ((x)*(x))

main()
{
```

```
    int x,y,z;
    double devs[33][17];
    double sumdevs;

/* read in the stddevs for each class, feature */

   for (x=1; x<=32; x++)
      for (y=1; y<=16; y++)
 scanf("%lf",&devs[x][y]);

/*  compute and print the total deviation for each class */

   for (x=1; x<=32; x++)
       {
 sumdevs = 0.0;
 for (z=1; z<=16; z++)
            sumdevs += SQR(devs[x][z]);
         printf("%5.1f is total std dev for class %3d \n",sqrt(sumdevs),x);
        }

}  /* end of main */

double sqrt(num)
/* returns the square root using Newton's method */
double num;
{
 double guess;
 guess = 100.0;
do
   guess = guess- (guess*guess-num)/(2.0*guess);
while (ABS(guess*guess-num) > 0.00000001);
return(guess);
}
```

D.11 repcrunch.c

```
/******** repcrunch.c ********************************/

/* program to analyze the item_report.
```

```
   Reads the edited report file giving the actual class and
   the guessed class by the neural network for  each test vector.  Produces
   output tables (confusion matrix).

The script file awkrep prepares an item_report file for input.
   Std input and output redirection to be used.
 ********************************************** */

#include <stdio.h>
#include <math.h>
#define TOPN  3

main()
{
    int x,y,z, class,guess,lines,classes,tabend;
    int mtx[32][32] ;
    float ops[31];

   for (x=0; x<=31; x++)
     for (y=0; y<=31; y++)
mtx[x][y] = 0;

   for (x=0; x<=31; x++)
      ops[x] = 0.0;

      scanf("%d %d ", &lines,&classes);
/* read in the report test data */

   for (z=1; z<=lines;z++)
   {
      scanf("%d %d ", &class,&guess);
      for (x=classes; x>=1; x--)
 scanf("%f", &ops[x]);

      mtx[class][guess]++ ;

      if (guess != class) /* ie there is an error */
       {
 mtx[class][0]++;   /* add one to error for class */
 if (ranking(class,ops) > TOPN)
    mtx[class][31]++ ;  /* incr topn error counter  */
```

```
        }

    }

/* print out the confusion mtx for all classes */

if (classes > 16)
     tabend = 15;
else tabend = classes;

  printf("Class");
  for (x=1; x<=tabend; x++)
       printf("\& %4d",x);
     printf("\\\\\n");
     printf("\n");

  for (y=1; y<=classes; y++)
     {
      printf("%6d",y);
      for (x=1; x<=tabend; x++)
   printf("\& %3d ",mtx[x][y]);
      printf("\\\\\n");
     }
  printf("Errors\n");
  for (x=1; x<=tabend; x++)
       printf("\& %3d",mtx[x][0]);
     printf("\\\\\n");

  printf("Exemplars not in top %3d\n",TOPN);
  for (x=1; x<=tabend; x++)
       printf("\& %3d",mtx[x][31]);
     printf("\\\\\n");
     printf("\n");

if (classes > 16)  /*Have to print a second table */
  {
  printf("Class");
  for (x=tabend+1; x<=classes; x++)
       printf("\& %4d",x);
     printf("\\\\\n");
     printf("\n");
```

```
  for (y=1; y<=classes; y++)
     {
      printf("%6d",y);
      for (x=tabend+1; x<=classes; x++)
   printf("\& %3d ",mtx[x][y]);
      printf("\\\\\n");
     }
  printf("Errors\n");
  for (x=tabend+1; x<=classes; x++)
       printf("\& %3d",mtx[x][0]);
     printf("\\\\\n");

  printf("Exemplars not in top %3d\n",TOPN);
  for (x=tabend+1; x<=classes; x++)
       printf("\& %3d",mtx[x][31]);
     printf("\\\\\n");
     printf("\n");
  }

} /* end of main */

int ranking(class,outp)
/* RETURNS THE RANKING OF CLASS BASED ON OUTPUTS OF NET */
int class;
float outp[31];
{
  int rank,a;
  /* COUNT HOW MANY O/P'S ARE >= THE CLASS OUTPUT */
  rank = 1;
  for (a=1; a<=30; a++)
    if ((a != class) && (outp[a] >= outp[class]))
      rank++;
  return (rank);
}
```

D.12 awkrep

```
   % awkrep
   % Script file to prepare the item_report output by
   % Neural Graphics for the analysis program repcrunch.c
```

```
% Can do multiple files at once
%
for file in $*
do
  awk '/Item/{print $4+1,$6+1,$7,$8,$9,$10,$11,$12,$13,$14,$15,$16,$17,
 $18,$19,$20,$21,$22}' $file > $file.1
  awk '/Total/{print $2+1,$4,$6,$8}' $file > x$file
  wc -l x$file > $file.3
  awk '{print $1}' $file.3 > $file.cl
  wc -l $file.1 > $file.4
  awk '{print $1}' $file.4 > $file.lc
  cat $file.lc $file.cl $file.1 >$file
  rm $file.*
  echo 'File is ready for repcrunch.c!'
done
```

Bibliography

1. Brown, Joe R. and others. "Comparison of two neural net classifiers to a quadratic classifier for millimeter wave radar", *Proceedings of SPIE.* 1294:217-224. Bellingham, WA: SPIE-The International Society for Optical Engineering, 1990.
2. Cybenko, G. "Approximation of Superpositions of a Sigmoidal Function", *Mathematics of Control, Signals and Systems*, 2:303-314 (March, 1989).
3. Ersoy, O. K. and D. Hong. " Parallel, Self-Organizing, Hierarchical Neural Networks", *IEEE Transactions on Neural Networks*, 1:167-178 (June 1990).
4. Gnanadesikan, R. and J. R. Kettenring. "Discriminant Analysis and Clustering", *Statistical Science*, 4:34-69 (1989).
5. Howitt, Ivan. "Radar warning receiver emitter identification processing utilizing artificial neural networks", *Proceedings of SPIE.* 1294:211-216. Bellingham, WA: SPIE-The International Society for Optical Engineering, 1990.
6. Intrator, Nathan. "A neural network for feature extraction", *Advances in Neural Information Processing Systems 2.* 719-724. Denver, CO: Morgan Kaufmann Publishers, 1990.
7. Kuhl, F.P., A.P. Reeves, and R.J. Prokop. *A Neural Network Object Recognition System*, July, 1990. Contract ARFSD-TR-90008 U.S. Army Research Office, Research Triangle Park, NC (AD A225 627).
8. Lin, Wei-Chung and others. "A Hierarchical Multiple View Approach to 3-Dimensional Object Recognition", *IEEE Transactions on Neural Networks.* 2:84-92 (January 1991).
9. Lippmann, Richard P. "Pattern Classification Using Neural Networks," *IEEE Communications Magazine*, 2:47-63 (December 1990).
10. Rogers, Steven K. and others. *An Introduction to Biological and Artificial Neural Networks.* Wright-Patterson AFB. OH: Air Force Institute of Technology. October 23, 1990.
11. Roth, M.W. "Neural Network Technology for Automatic Pattern Recognition". *IEEE Transactions on Neural Networks*, 1:32-38, (March 1990).
12. Ruck, Dennis W. and others. "The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function", *IEEE Transactions on Neural Networks*, 1:296-298 (December 1990).
13. Ruck, Capt Dennis W. *Characterization of Multilayer Perceptrons and their Application to Multisensor Automatic Target Detection.* PhD dissertation. School of Engineering. Air Force Institute of Technology (AU). Wright-Patterson AFB OH. December 1990 (AD-A229 035).

14. Sun, G. Z., H. H. Chen and Y. C. Lee. "Parallel Sequential Induction Network: A New Paradigm of Neural Network Architecture", *Proceedings of the IEEE International Joint Conference on Neural Networks.* 489-496. San Diego: IEEE Press, 1988.

15. Tarr, Gregory L., and others. "AFIT Neural Network Development Tools and Techniques for Modeling Artificial Neural Networks", *Proceedings of SPIE.* 1294:211-216. Bellingham, WA: SPIE-The International Society for Optical Engineering, 1990.

16. Villa, Mark F. and Kevin D. Reilly. "Hierarchical Neural Networks", *SPIE Proceedings of the Second Workshop on Neural Networks.* 1515:657-664. San Diego: The Society for Computer Simulation, International, 1991.

17. Willson, Gregory B. "Radar classification using a neural network", *Proceedings of SPIE.* 1294:200-210. Bellingham. WA: SPIE-The International Society for Optical Engineering, 1990.

18. Zahirniak, Daniel R., *Characterization of Radar Signals Using Neural Networks.* Master's thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990 (AD-A230 582).

December 1991 Master's Thesis

RADAR SYSTEM CLASSIFICATION USING NEURAL NETWORKS

David M. Cameron, Captain, CAF

Air Force Institute of Technology, WPAFB OH 45433-6583

AFIT/GSO/ENS/91D-03

This study investigated methods of improving the accuracy of neural networks in the classification of large numbers of classes. A literature search revealed that neural networks have been successful in the radar classification problem, and that many complex problems have been solved using systems of multiple neural networks. The experiments conducted were based on 32 classes of radar system data. The neural networks were modelled using a program called the *Neural Graphics Analysis System*. It was found that the accuracy of the individual neural networks could be increased by controlling the number of hidden nodes, the relative numbers of training vectors per class, and the number of training iterations. The maximum classification accuracy of 96.8% was achieved using a hierarchy of neural networks in which the classes were partitioned based on their performances in a large neural network trained with all classes.

Artificial Neural Networks, Radar Warning Receiver 208

Unclassified Unclassified Unclassified UL